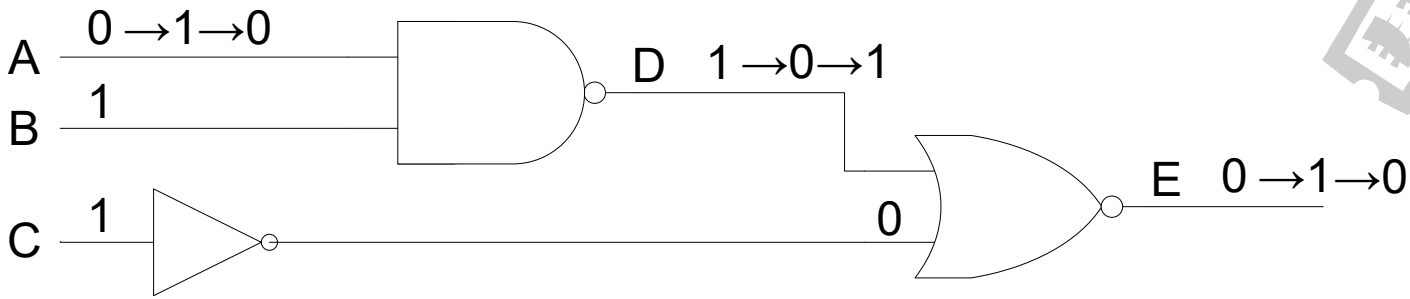
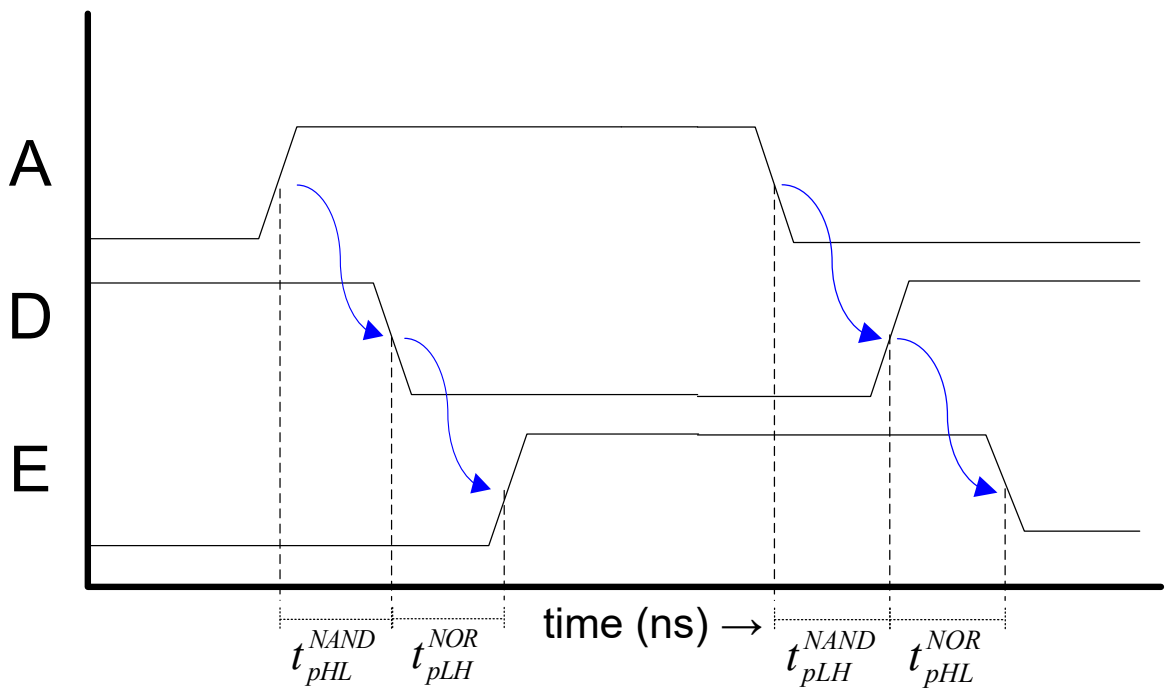
UM EECS 270 F22
Introduction to Logic Design

23. Review

Timing Diagrams





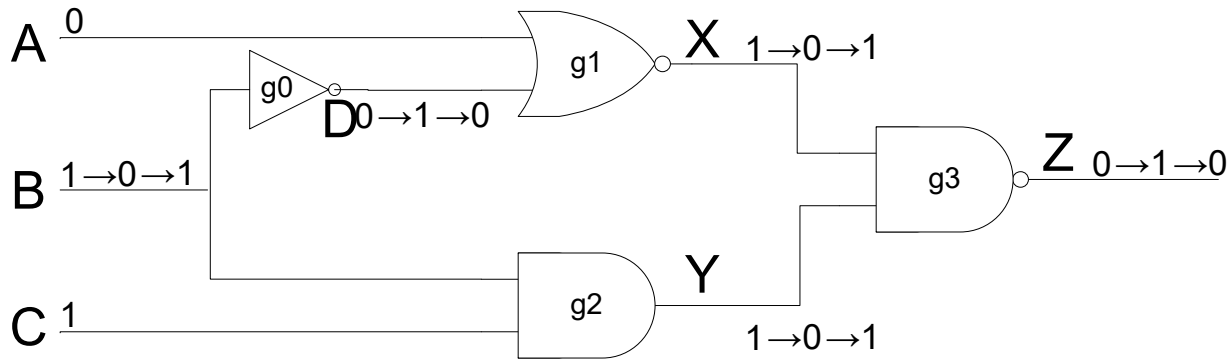
Causality Arrow

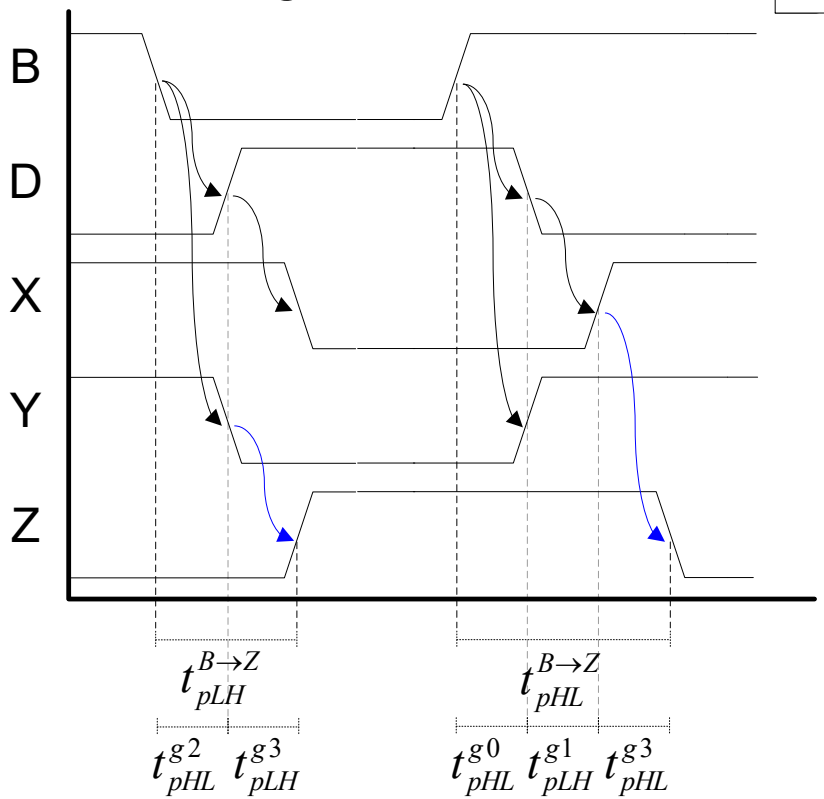
Gate Propagation Delay

t_{pHL}^{NAND}

gives NAND gate propagation delay from input to output when <u>output</u> is changing from H to L

Multiple paths from input to output





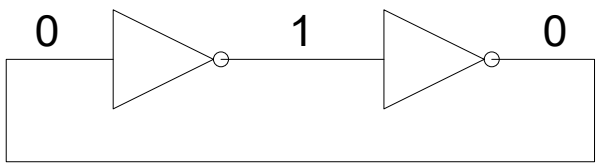
Assume all rising and falling gate delays are 1ns:

$$t_{pLH}^{B\to Z} = t_{pHL}^{g2} + t_{pLH}^{g3} = 2ns$$

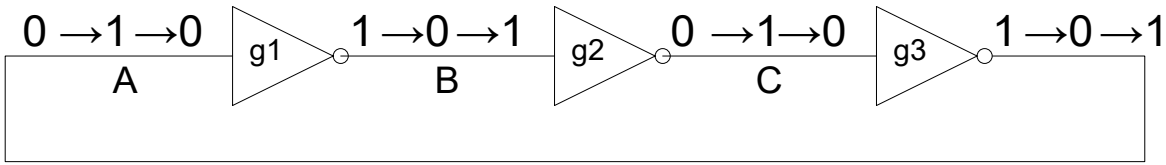
$$t_{pHL}^{B\to Z} = t_{pHL}^{g0} + t_{pLH}^{g1} + t_{pHL}^{g3} = 3ns$$

The overall circuit delay depends on the transition path taken from the inputs to the output

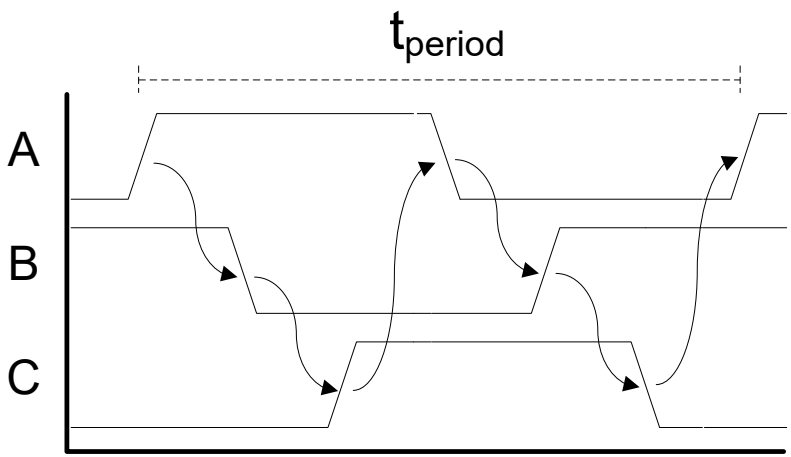
Bi-Stable Circuit & Ring Oscillator



This circuit is **bi-stable**



This circuit is known as a **ring-oscillator**



$$t_{period} = t_{pHL}^{g1} + t_{pLH}^{g2} + t_{pHL}^{g3} + t_{pLH}^{g1} + t_{pHL}^{g2} + t_{pLH}^{g3}$$

Assume all gate delays are 1ns:

$$t_{period} = 6ns \qquad f_{osc} = \frac{1}{6ns} = 166MHz$$

Formal Definition of Switching Algebra

- Base set: B_2={0, 1}
- One unary operation: NOT or COMPLEMENT: $(x', \bar{x}, \neg x)$
- Two binary operations: AND $(\cdot,\ \wedge)$, OR $(+,\ \vee)$
- Postulates (axioms):

Postulate	Defines	A	B
P1	Switching Variables	$x = 0$ iff $x \neq 1$	$x = 1$ iff $x \neq 0$
P2	NOT	$0'=1$	$1'= 0$
P3		$0 \cdot 0 = 0$	$1 + 1 = 1$
P4	AND / OR	$1 \cdot 1 = 1$	$0 + 0 = 0$
P5		$0 \cdot 1 = 1 \cdot 0 = 0$	$0 + 1 = 1 + 0 = 1$

- Duality: $0 \leftrightarrow 1, \cdot \leftrightarrow\ +$

(Some) Theorems

	A	Name	B
T1	$x \cdot 1 = x$	Identities	$x + 0 = x$
T2	$x \cdot 0 = 0$	Null Elements	$x + 1 = 1$
T3	$x \cdot x = x$	Idempotency	$x + x = x$
T4		Involution $(x')' = x$	
T5	$x \cdot x' = 0$	Complements	$x + x' = 1$
T6	$x \cdot y = y \cdot x$	Commutativity	$x + y = y + x$
T7	$x \cdot (x + y) = x$	Absorption	$x + (x \cdot y) = x$
T8	$x \cdot (x' + y) = x \cdot y$	No Name	$x + (x' \cdot y) = x + y$
T9	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	Associativity	$(x + y) + z = x + (y + z)$
T10	$x \cdot (y + z) = x \cdot y + x \cdot z$	Distributivity	$x + (y \cdot z) = (x + y) \cdot (x + z)$
T11	$x \cdot y + x' \cdot z + y \cdot z$ $= x \cdot y + x' \cdot z$	Consensus	$(x + y) \cdot (x' + z) \cdot (y + z)$ $= (x + y) \cdot (x' + z)$
T12	De Morgan's $f(x_1, \ldots, x_n, 0, 1, \cdot, +)' = f(x_1', \ldots, x_n', 1, 0, +, \cdot)$		

Switching Functions

- $f(x_1, x_2, \ldots, x_n)$ is a mapping from $B_2^n \rightarrow B_2$
- *f* can be specified by many equivalent expressions or by tables of combinations (truth tables)
- Elementary functions:
  - A minterm m_i is an AND term of *n* literals
  - A maxterm M_i is an OR term of *n* literals

  Ex: 4 variables *A*, *B*, *C*, *D*
    - $m_5(A, B, C, D) = A'BC'D \qquad (0101)$
    - $M_5(A, B, C, D) = A + B' + C + D' \qquad (0101)$
- $m_i = 1$ for exactly one combination of variables, and 0 for all others
- $M_i = 0$ for exactly one combination of variables, and 1 for all others
- $m_i = M_i'$

Canonical Forms

- Canonical Sum-of-Products (SOP)
  - Also known as Disjunctive Normal Form (DNF)
  - Sum of minterms (those for which $f = 1$)
  - Shorthand: $\sum(\ldots)$
- Canonical Product-of-Sums (POS)
  - Also known as Conjunctive Normal Form (CNF)
  - Product of maxterms (those for which $f = 0$)
  - Shorthand: $\prod(\ldots)$

Canonical Forms

- Canonical Sum-of-Products (SOP)
  - Also known as Disjunctive Normal Form (DNF)
  - Sum of minterms (those for which *f* = 1)
  - Shorthand: $\sum(\ldots)$
- Canonical Product-of-Sums (POS)
  - Also known as Conjunctive Normal Form (CNF)
  - Product of maxterms (those for which *f* = 0)
  - Shorthand: $\prod(\ldots)$

Decimal	*x y z*	*f*
0	000	1
1	001	0
2	010	1
3	011	1
4	100	0
5	101	0
6	110	1
7	111	1

$$f(x,y,z) = \sum_{x,y,z}(0,2,3,6,7)$$

$$f(x,y,z) = x'y'z' + x'yz' + x'yz + xyz' + xyz$$

$$f(x,y,z) = \prod_{x,y,z}(1,4,5)$$

$$f(x,y,z) = (x+y+z')(x'+y+z)(x'+y+z')$$

Don't Cares

Decimal	$x\,y\,z$	f
0	000	1
1	001	0
2	010	d
3	011	1
4	100	0
5	101	d
6	110	1
7	111	1

$$f(x,y,z) = \sum_{x,y,z}(0,3,6,7) + d(2,5)$$

$$f(x,y,z) = \prod_{x,y,z}(1,4)d(2,5)$$

On-Set = {0, 3, 6, 7}
Off-Set = {1, 4}
Don't-Care-Set = {2, 5}

Code Word Representation of Product Terms

Variables: $u\ v\ w\ x\ y\ z$

Code Word	Product Term	#minterms "covered"
001101	$u'v'wxy'z$	$1\ \{m_{13}\}$
0-1-01	$u'wy'z$	$2^2 = 4\ \{m_9, m_{13}, m_{25}, m_{29}\}$

$$\#\text{minterms covered by code word} = 2^{\#\text{missing literals}}$$

Boole's (Shannon's) Expansion Theorem

- Decomposition of a switching function of *n* variables into functions of $n-1$ variables
  - $f(x_1, x_2, \dots, x_n) = x_1' f(0, x_2, \dots, x_n) + x_1 f(1, x_2, \dots, x_n)$
  - $f(x_1, x_2, \dots, x_n) = [x_1' + f(1, x_2, \dots, x_n)][x_1 + f(0, x_2, \dots, x_n)]$
- The functions resulting from fixing x are referred to as co-factors
  - $f(0, x_2, \dots, x_n)$ is the negative cofactor of f wrt x_1
  - $f(1, x_2, \dots, x_n)$ is the positive cofactor of f wrt x_1
- Notation:
  - $f_{x_1'} = f(0, x_2, \dots, x_n)$
  - $f_{x_1} = f(1, x_2, \dots, x_n)$

Switching Functions of 2 Variables

$$f(x, y) = a_0 m_0 + a_1 m_1 + a_2 m_2 + a_3 m_3$$

$a_3 a_2 a_1 a_0$	$f(x, y)$	Name	Symbol	Unique?
0000	0	Inconsistency		‡
0001	$x'y'$	NOR	$x \downarrow y$	
0010	$x'y$	Inhibition		
0011	x'	NOT		
0100	xy'	Inhibition		
0101	y'	NOT		
0110	$x'y + xy'$	XOR	$x \oplus y$	‡
0111	$x' + y'$	NAND	$x \uparrow y$	
1000	xy	AND	$x \cdot y = x \wedge y = xy$	‡
1001	$xy + x'y'$	XNOR / EQV	$x \odot y$	
1010	y	Transfer		
1011	$x' + y$	Implication	$x \rightarrow y$	
1100	x	Transfer		‡
1101	$x + y'$	Implication	$y \rightarrow x$	
1110	$x + y$	OR	$x + y = x \vee y$	‡
1111	1	Tautology		‡

Properties of XOR (modulo-2 Addition)

- Commutativity: $x \oplus y = y \oplus x$
- Associativity: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- Distributivity: $x(y \oplus z) = xy \oplus xz$
- Relationship to XNOR: $(x \oplus y)' = x \odot y$
  - XNOR is "equal"
  - XOR is "not equal"
- Conditional Complementation:

$$s \oplus x = \begin{cases} x & \text{if } s = 0 \\ x' & \text{if } s = 1 \end{cases}$$

- Parity: Value of $f = x_1 \oplus x_2 \oplus \ldots \oplus x_n$
  - Remains unchanged if an even number of variables are complemented
  - Is complemented if an odd number of variables are complemented
- Any identity $f(X) = g(X)$ can be re-expressed as $f(X) \oplus g(X) = 0$

Functional Completeness

- A set of operations is functionally-complete (or universal) iff every switching function can be expressed entirely by means of operations from this set
- The following are functionally-complete operation sets
  - $\{+,\cdot,'\}$ (by definition)
  - $\{+,'\}$ (by De Morgan's theorem)
  - $\{\cdot,'\}$ (by De Morgan's theorem)
  - {NAND}: $x \uparrow y = x' + y'$

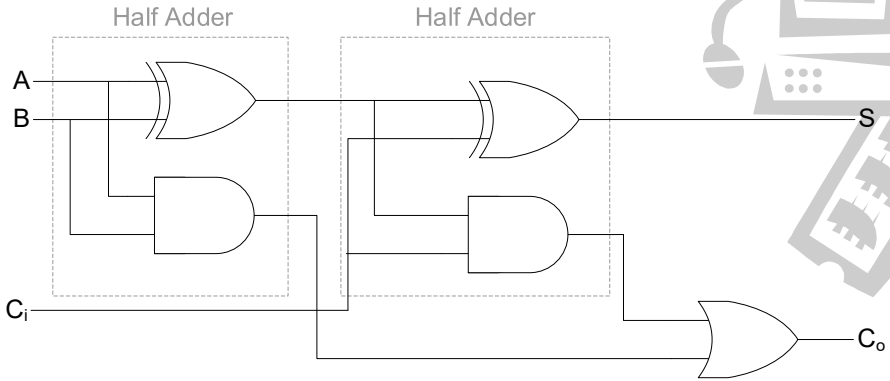
Representation of Numbers

- Positional
- Radix: 2, 4, 8, 10, 16 (and any other!)
- Unsigned
- Signed:
  - Signed Magnitude
  - Ones' Complement
  - Two's Complement

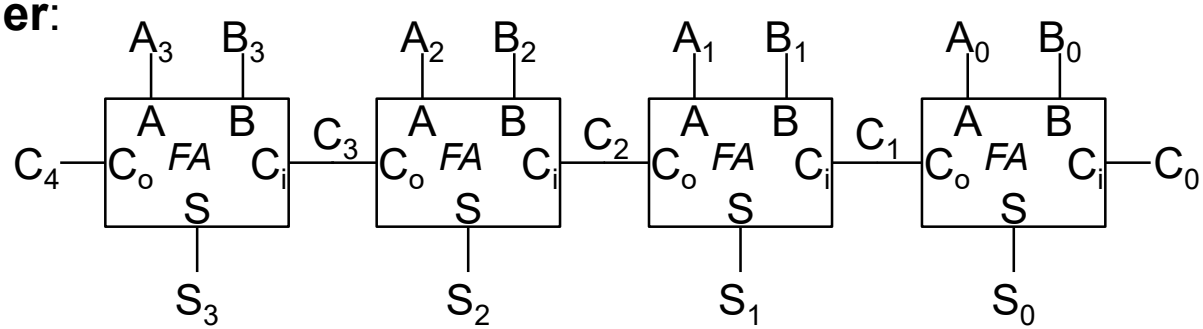
- Final Circuit:

$$S = A \oplus B \oplus C_i$$

$$C_o = AB + (A \oplus B)C_i$$



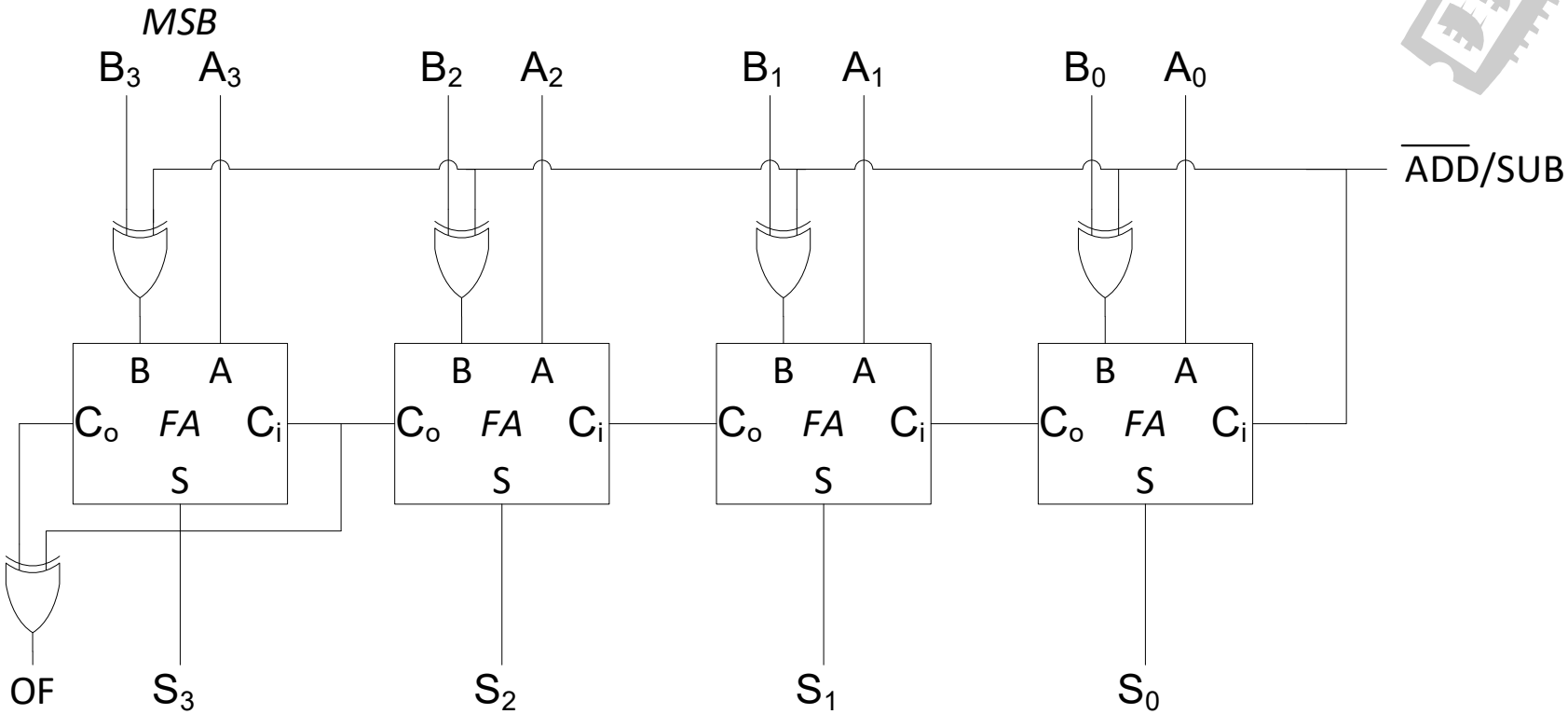
- This circuit is called a **Full Adder (FA)**
- After all that design work, we really just have 2 HAs with an OR gate
- To make an n-bit adder, simply cascade Full Adders to make a **Ripple Carry Adder**:



- What about subtraction hardware?
  - Could design subtractor hardware using process similar to adder design
  - Simpler way: re-use our addition hardware! A – B = A + (– B)

(– B)*??? How do we represent* ***negative numbers?***

Ripple-carry Adder/Subtractor with Overflow Detection



$\overline{ADD}/SUB = 0 \rightarrow S = A + B$

$\overline{ADD}/SUB = 1 \rightarrow S = A - B$

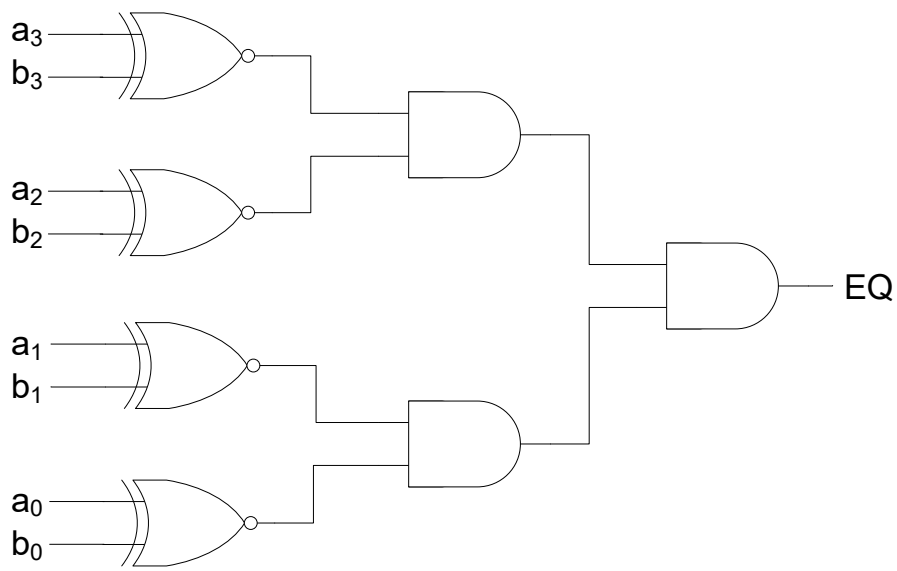
$OF = 1 \rightarrow$ Overflow

Equality Comparator

- Want a signal, EQ, that is 1 iff two 4-bit numbers, A and B, are equal. Under what logic conditions are A and B equal?
  - Equal if $a_3 = b_3$ AND $a_2 = b_2$ AND $a_1 = b_1$ AND $a_0 = b_0$
- Which gate performs an equality comparison?
  - XNOR!

A	B	A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

$$EQ = (a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot (a_0 \odot b_0)$$



How many gates are required to implement an *n*-bit "parallel" comparator?

Assume $n = 2^k$

#gates = $2^k + 2^{k-1} + 2^{k-1} + \ldots 2^0$

$= 2^{k+1} - 1$

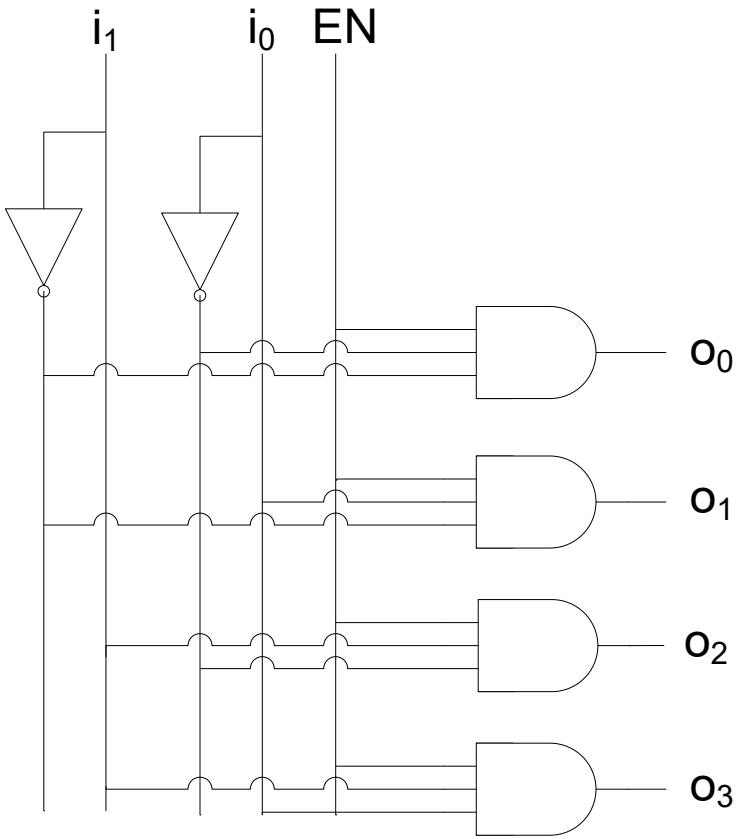
$= 2n - 1$

Magnitude Comparator

- Want to design a signal, AgrB, that is 1 iff A > B (A and B are 4-bit numbers). Under what logic conditions is A > B?
  - A > B if $a_3 = 1$ and $b_3 = 0$
  - A > B if $a_2 = 1$ and $b_2 = 0$ *and* $a_3 = b_3$
  - A > B if $a_1 = 1$ and $b_1 = 0$ and $a_2 = b_2$ and $a_3 = b_3$
  - A > B if $a_0 = 1$ and $b_0 = 0$ and $a_1 = b_1$ and $a_2 = b_2$ and $a_3 = b_3$

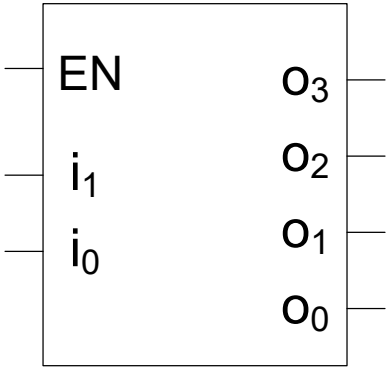
$$
\begin{aligned}
AgrB = a_3 \cdot \overline{b_3} + \\
&(a_3 \odot b_3) \cdot a_2 \cdot \overline{b_2} + \\
&(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot a_1 \cdot \overline{b_1} + \\
&(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot a_0 \cdot \overline{b_0}
\end{aligned}
$$

Decoder Circuit

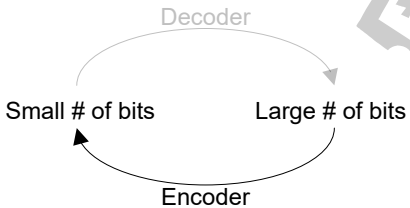


Symbol:

2-to-4 Decoder



Encoders



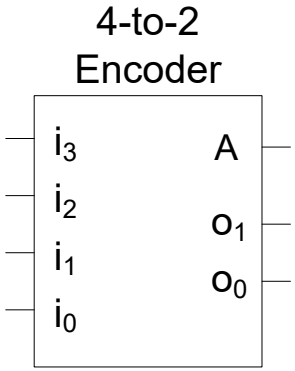
- Opposite of decoders: an encoder takes as input a code with a large number of bits and outputs a corresponding code with a small number of bits
- Most common encoder maps a 2^n one-hot code to an n-bit binary number, where the binary number represents the input number that is asserted
- What if all input signals are deasserted? Requires additional output, usually called A (active) that is asserted if *at least* one input is asserted

i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
all others				d	d	d

$$o_1 = i_3 + i_2$$

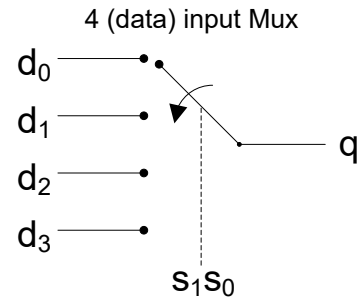
$$o_0 = i_3 + i_1$$

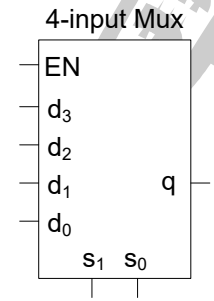
$$A = i_3 + i_2 + i_1 + i_0$$



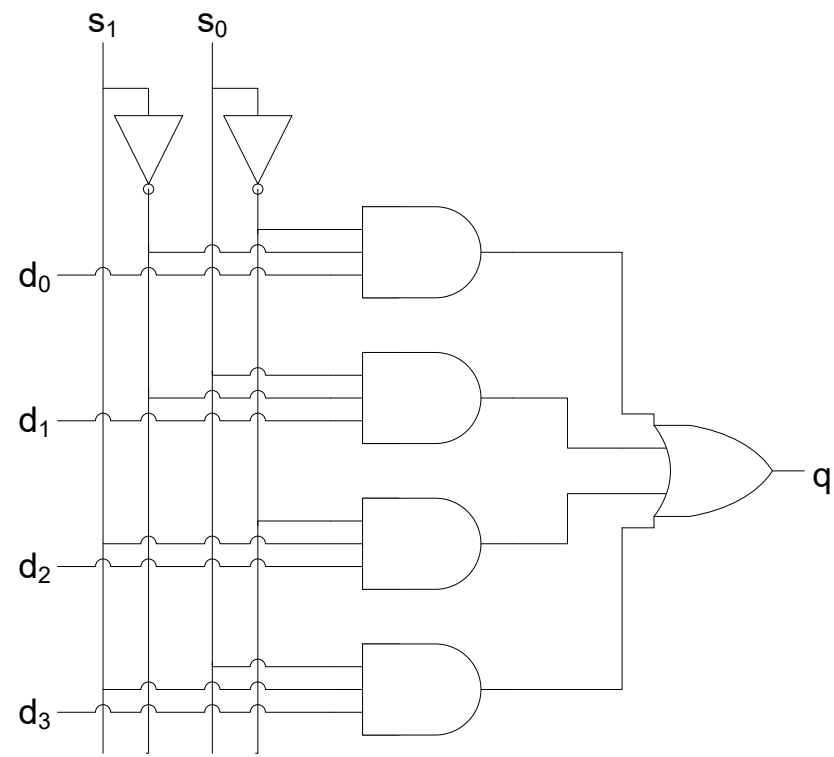
Multiplexer (Mux)

- A **mux** is a digital switch
- The output copies one of *n* data inputs, depending on the value of the *select inputs*
- Implementation:





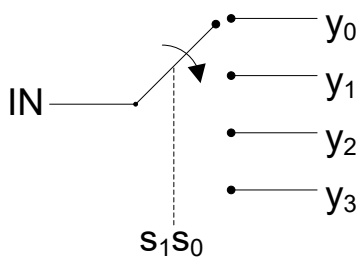
$$q = s_1' s_0' d_0 + s' s_0 d_1 + s_1 s_0' d_2 + s_1 s_0 d_3$$



Product terms are mutually exclusive!

Demultiplexer (demux)

- A **demux** connects an input signal to one of several output signals, depending on the value of the select signals

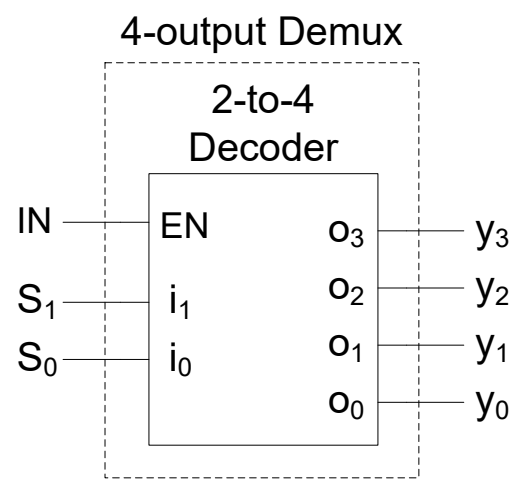


s_1	s_0	y_0	y_1	y_2	y_3
0	0	IN	0	0	0
0	1	0	IN	0	0
1	0	0	0	IN	0
1	1	0	0	0	IN

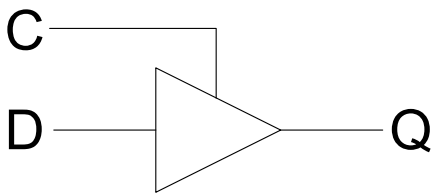
2-to-4 Decoder

EN	i_1	i_0	o_0	o_1	o_2	o_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

- How to implement?
  - TT should look very familiar…
  - Use a decoder!

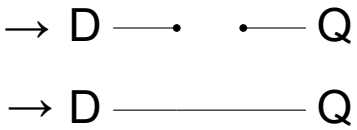


Tri-State Gates

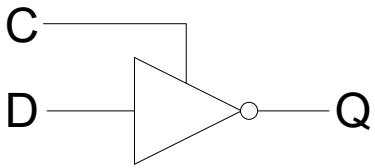


Tri-state Buffer

C	Q
0	High Impedance
1	D

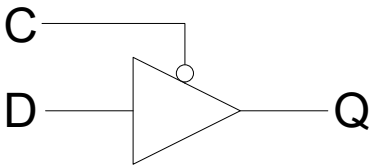


Other tri-state devices:



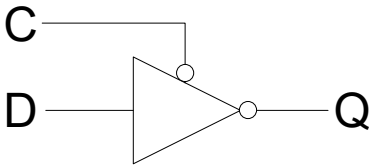
Tri-state Inverter

C	Q
0	High Impedance
1	$\overline{D}$



Active low
Tri-state Buffer

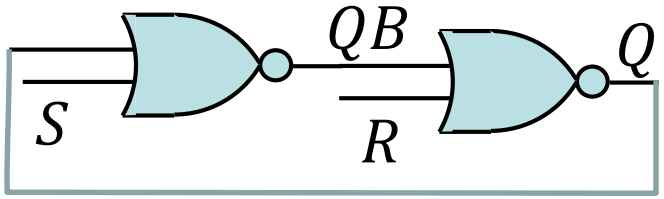
C	Q
0	D
1	High Impedance



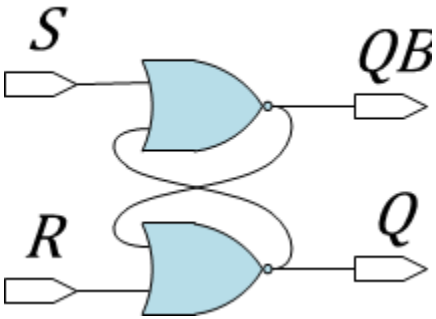
Active low
Tri-state Inverter

C	Q
0	$\overline{D}$
1	High Impedance

SR (Set-Reset) Latch

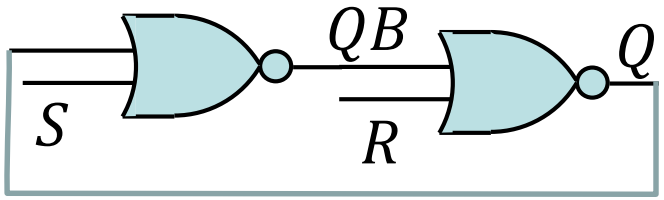


S	R	Q^+	QB^+	Function
0	0	Q	QB	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Invalid

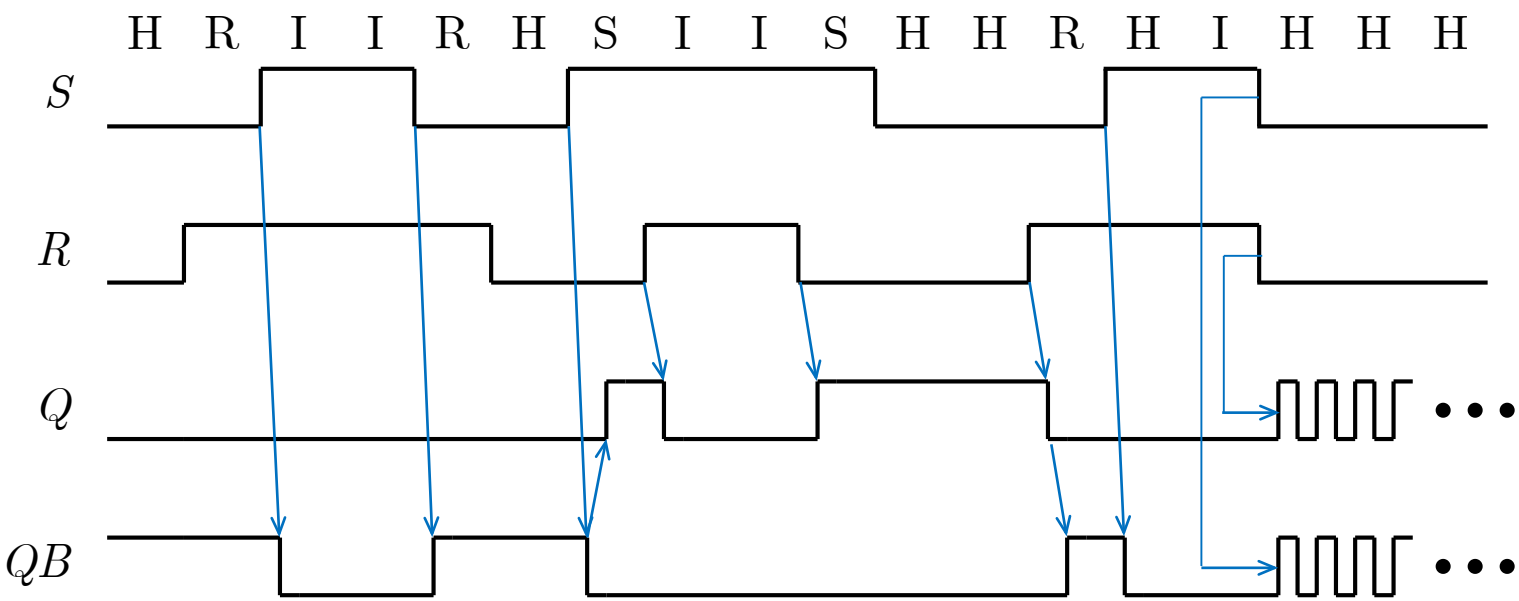


Cross-Coupled NOR Gates

SR Latch Timing

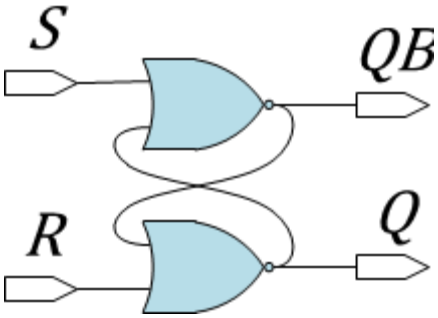


S	R	Q^+	QB^+	Function
0	0	Q	QB	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Invalid

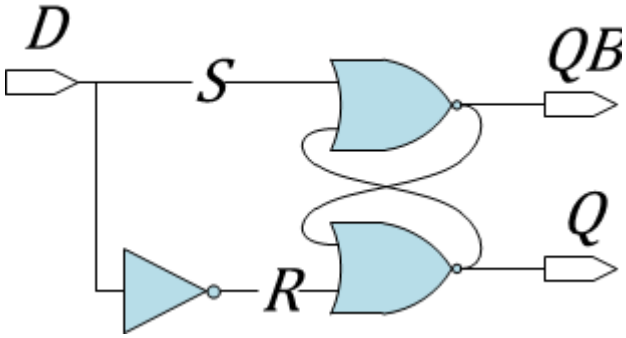


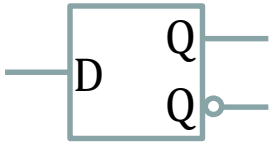
Latch Names and Symbols

SR Latch

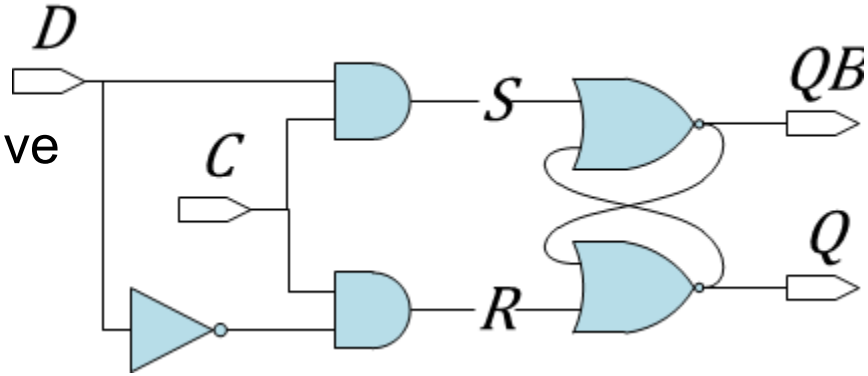


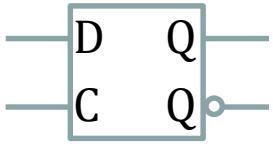
D Latch





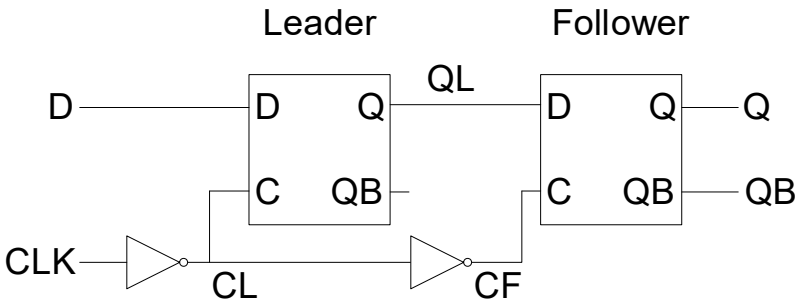
(Positive) Level Sensitive D Latch



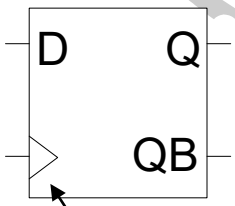


Edge-Triggered D Flip-flop

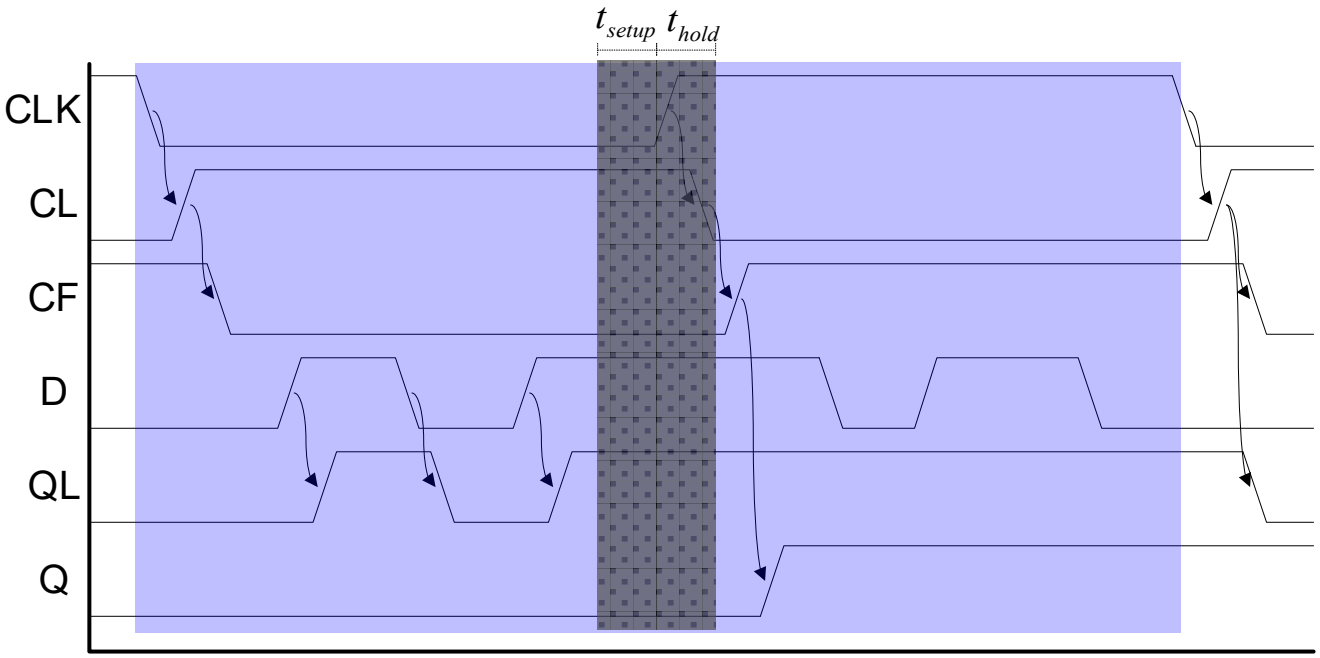
- Want a device that will sample its inputs *only once per clock cycle*



D	CLK	Q	QB
0	↑	0	1
1	↑	1	0
x	0	Last Q	Last QB
x	1	Last Q	Last QB

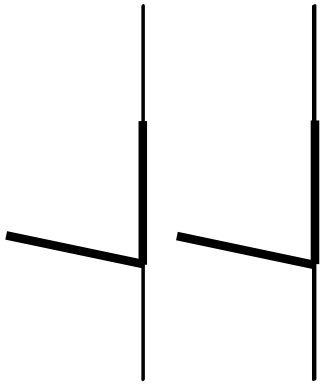


Indicates edge-trigged behavior



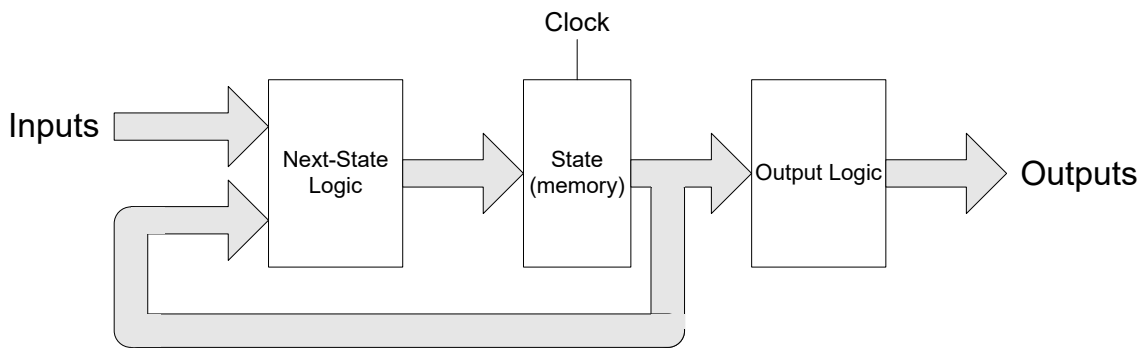
Leader transparent

Follower transparent

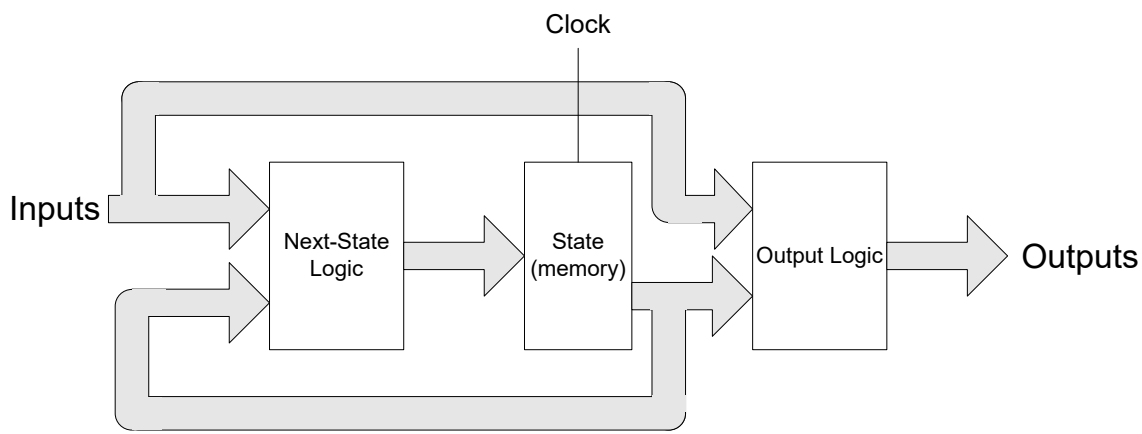


"Double-door" analogy

- Sequential Circuit Components:
  - **Next state logic** *(combinational)*: next state = f(current state, inputs)
  - **Memory** *(sequential)*: stores state in terms of state variables
  - **Output logic** *(combinational)*:
    - **<u>Moore Output</u>**: output = g(current state)

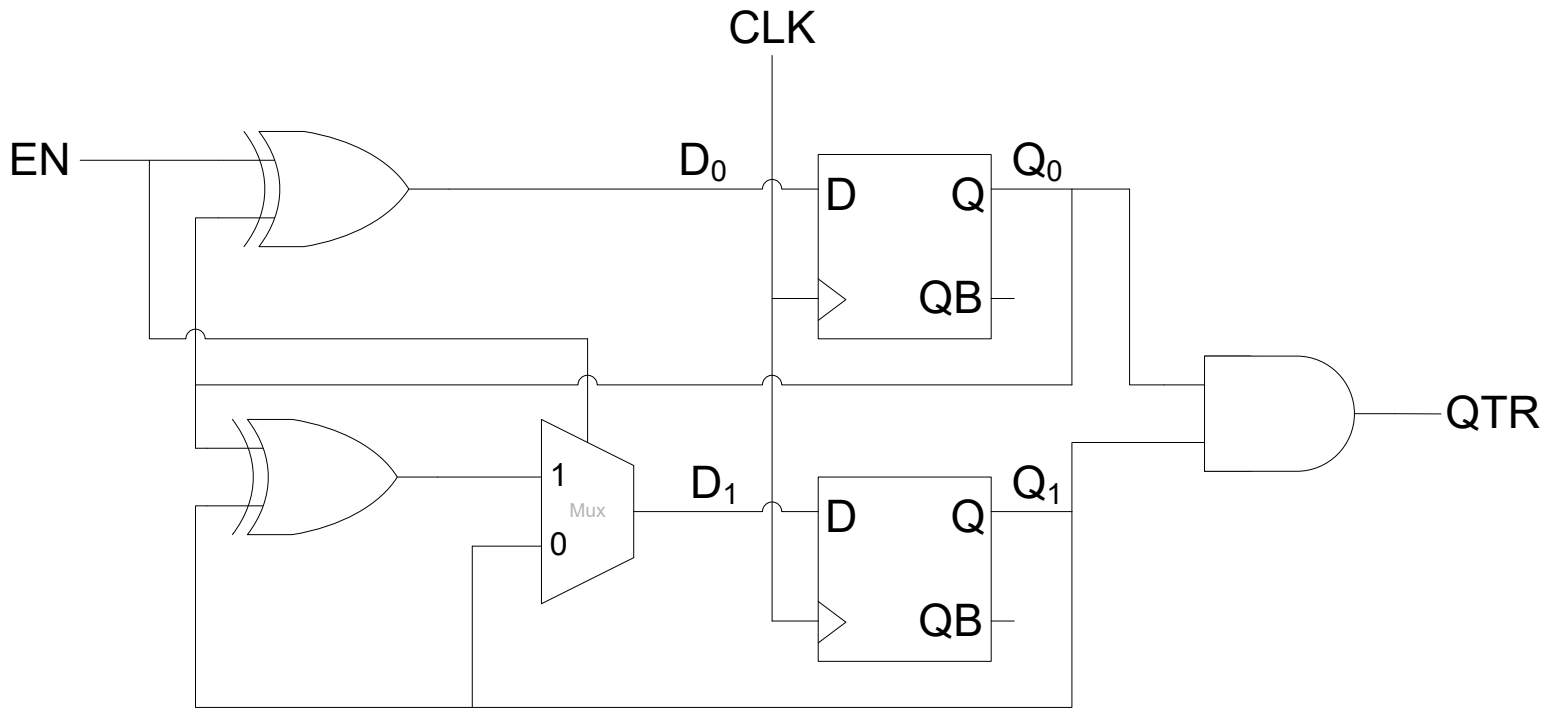


    - **<u>Mealy Output</u>**: output= g(current state, inputs)

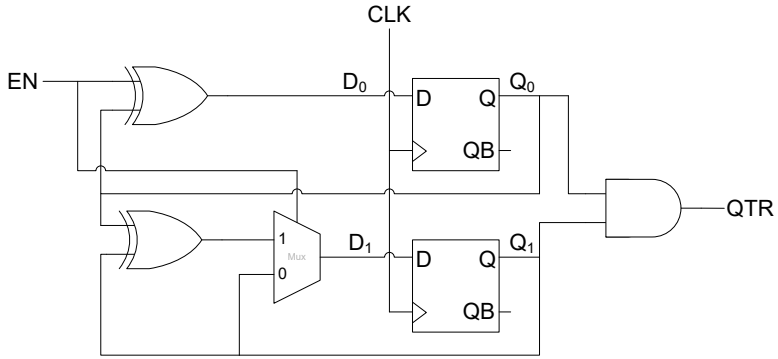


Sequential Circuit Analysis

- Goal: Given a sequential circuit, describe the circuit's behavior



- The **transition/output table** shows the next state and output for every current state/input combination
  - Entries of the table are obtained from the transition equations and the output equations



Transition Equations:

$$Q_0^+ = D_0 = EN \oplus Q_0$$

$$Q_1^+ = D_1 = EN \cdot (Q_0 \oplus Q_1) + \overline{EN} \cdot Q_1$$

Output Equation:

$$QTR = Q_0 \cdot Q_1$$

Transition/Output Table:

current state		*input* EN		*output*
Q_1	Q_0	0	1	QTR
0	0	00	01	0
0	1	01	10	0
1	0	10	11	0
1	1	11	00	1

$Q_1^+ \; Q_0^+$

next state

- **State labels** are a one-to-one mapping from state encodings to state names

Q_1	Q_0	State name
0	0	A
0	1	B
1	0	C
1	1	D

- The **state/output table** has the same format as the transition table, but state names are substituted in for state encodings

Transition/Output Table:

		EN		
Q_1	Q_0	0	1	QTR
0	0	00	01	0
0	1	01	10	0
1	0	10	11	0
1	1	11	00	1
		Q_1^+	Q_0^+	

State Assignments

Q_1	Q_0	State name
0	0	A
0	1	B
1	0	C
1	1	D

State/Output Table:

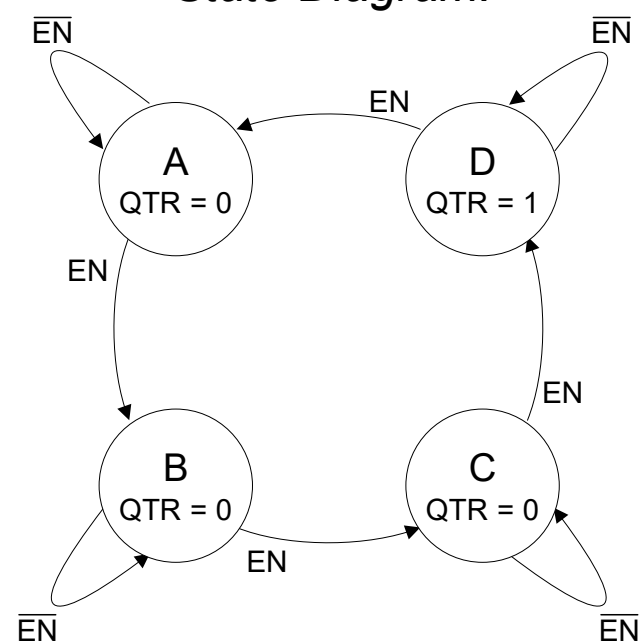
	EN		
S	0	1	QTR
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1
	S^+		

- A **state diagram** is a graphical representation of the information in the state/output table
- **Nodes** (or vertices) represent states
  - Moore machines: output values are written in state node
- **Arcs** (or edges) represent state transitions
  - Labeled with a **transition expression**
    - when an arc's transition expression evaluates to 1 for a given input combination, that arc is followed to the next state
  - Mealy machines: output values (or expressions) are written on arcs

State/Output Table:

	EN		
S	0	1	QTR
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1
	S^+		

State Diagram:



Sequential Design Example

- Problem description: design a Moore sequential circuit with one input IN and one output OUT, such that OUT is one iff IN is 1 for three consecutive clock cycles
- State table:

	IN		
S	0	1	OUT
zero1s	zero1s	one1	0
one1	zero1s	two1s	0
two1s	zero1s	three1s	0
three1s	zero1s	three1s	1
	S^+		

- State Assignments
  - What is the minimum number of state variables needed to encode four states?

$$2$$

  - In general, if we have *n* states, what is the minimum number of state variables needed to encode those states?

$$\lceil \log_2 n \rceil$$

State name	Q_1	Q_0
zero1s	0	0
one1	0	1
two1s	1	0
three1s	1	1

These state assignments may seem rather arbitrary – that's because they are! We will soon see the impact that state assignments have on our final circuit…

- Transition/output table

State/Output Table:

S	IN 0	IN 1	OUT
zero1s	zero1s	one1	0
one1	zero1s	two1s	0
two1s	zero1s	three1s	0
three1s	zero1s	three1s	1
	S^+		

State Assignments

State name	Q_1	Q_0
zero1s	0	0
one1	0	1
two1s	1	0
three1s	1	1

Transition/Output Table:

Q_1	Q_0	IN 0	IN 1	OUT
0	0	00	01	0
0	1	00	10	0
1	0	00	11	0
1	1	00	11	1
		Q_1^+	Q_0^+	

- Choose FF type:
  - Using D flip-flops will simplify things (as we'll see below…)

- Excitation table
  - Shows *FF input values required to create next state values* for every current state/input combination
  - If we're designing with D FFs, entries in excitation/output table are the same as those in transition/output table!
    - Because of D FF characteristic equation: $Q^+ = D$

Q_1	Q_0	IN 0	IN 1	OUT
0	0	00	01	0
0	1	00	10	0
1	0	00	11	0
1	1	00	11	1
		D_1	D_0	

- Excitation Logic

Excitation/Output Table:

		IN		
Q_1	Q_0	0	1	OUT
0	0	00	01	0
0	1	00	10	0
1	0	00	11	0
1	1	00	11	1
		D_1	D_0	

$$D_1 = IN' \cdot 0 + IN \cdot (Q_1' \cdot Q_0')'$$
$$= IN \cdot (Q_1 + Q_0)$$

$$D_0 = IN' \cdot 0 + IN \cdot (Q_1' \cdot Q_0)'$$
$$= IN \cdot (Q_1 + Q_0')$$

- Output Logic

$$OUT = Q_1 \cdot Q_0$$

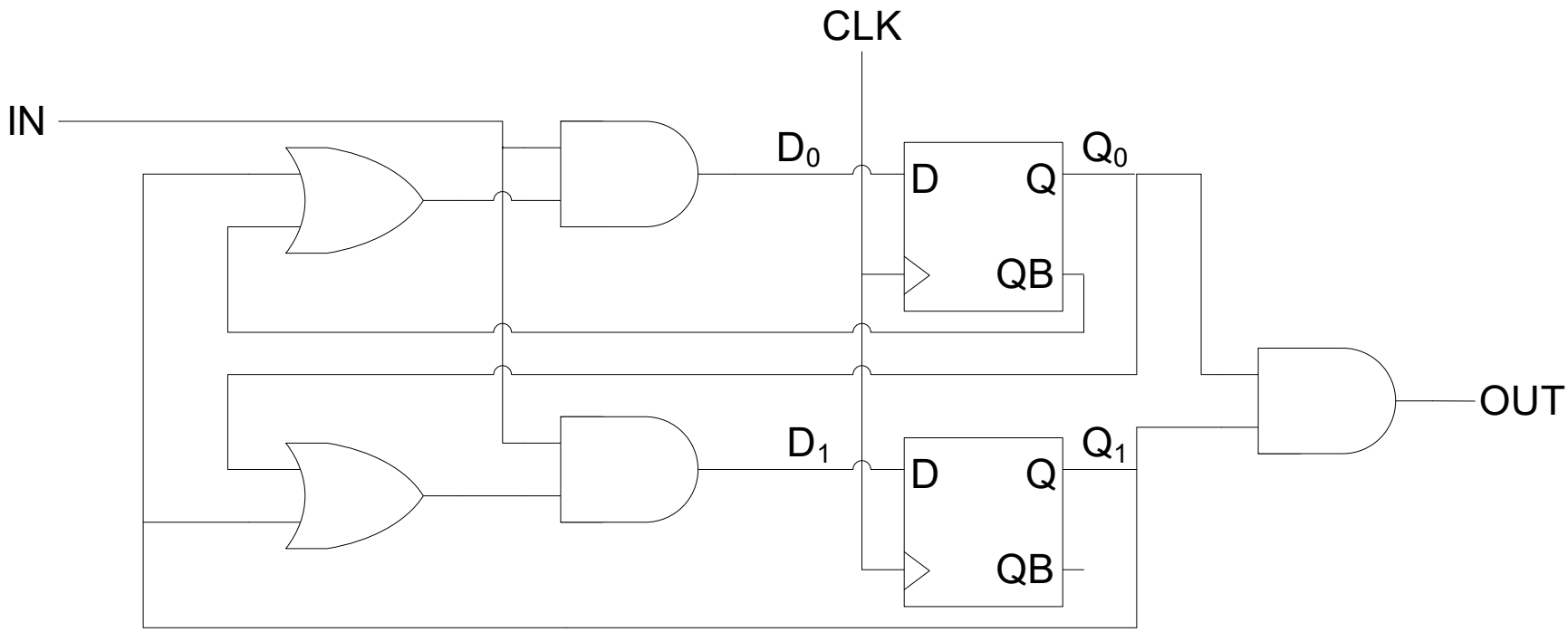
- Circuit:

Excitation Equations:

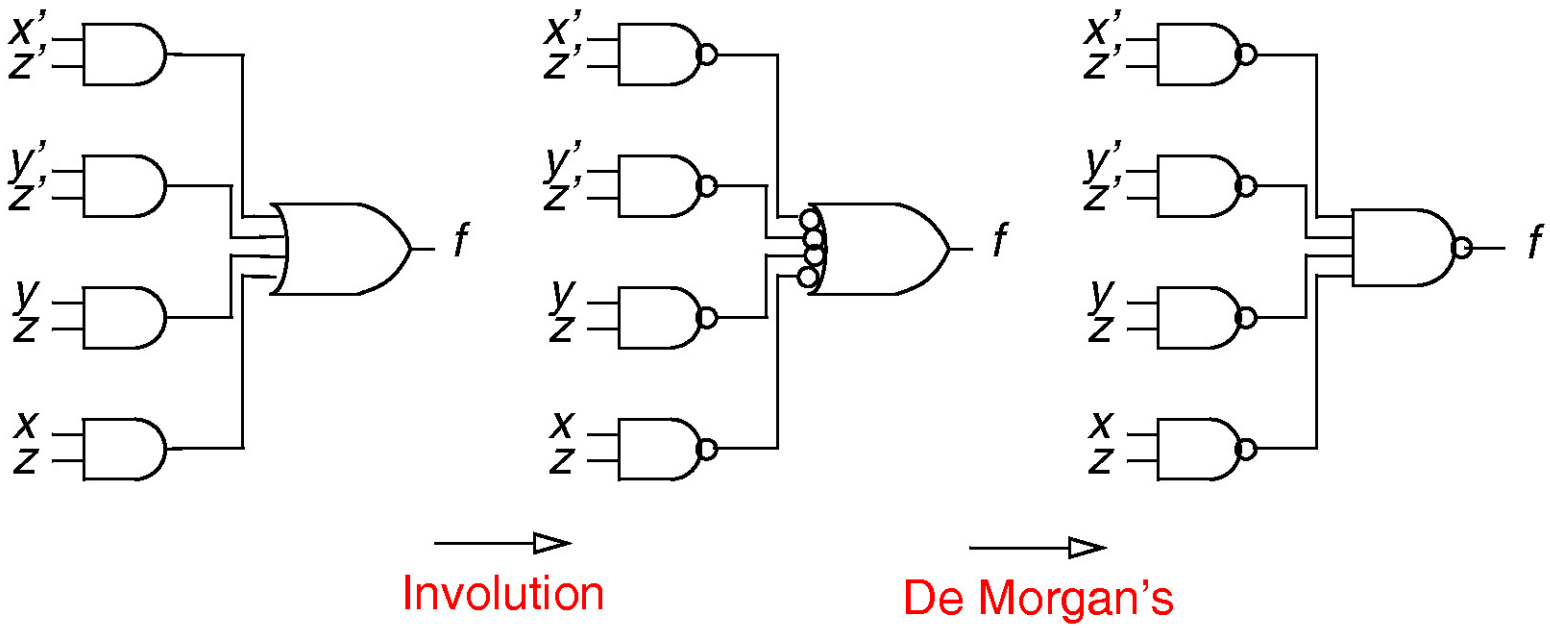
$$D_1 = IN \cdot (Q_1 + Q_0)$$
$$D_0 = IN \cdot (Q_1 + Q_0')$$

Output Equation:

$$OUT = Q_1 \cdot Q_0$$



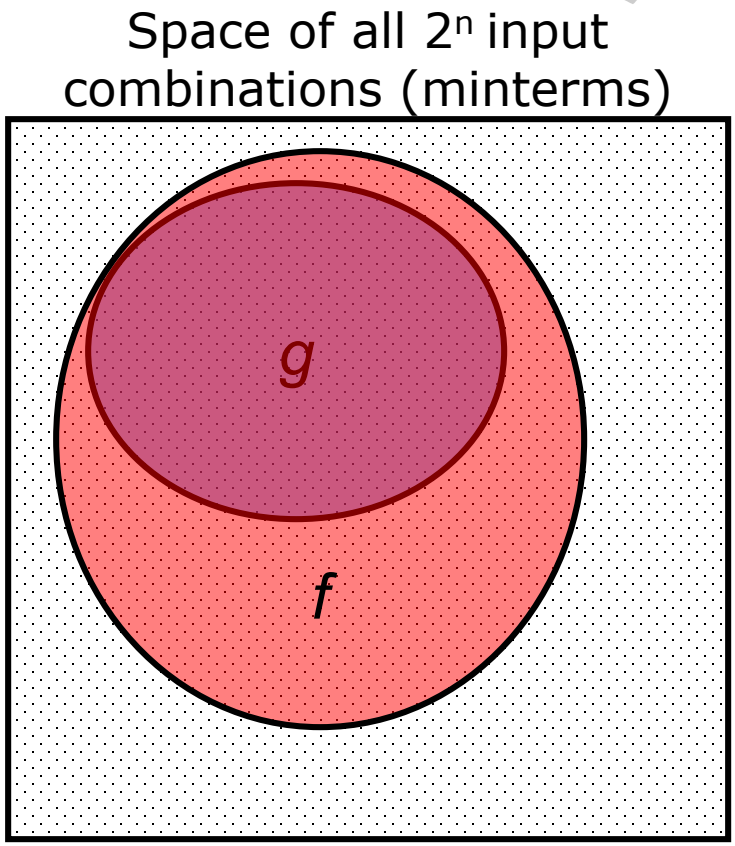
AND/OR ⇔ NAND/NAND



Covering

= input combinations for which *g* outputs 1

= input combinations for which *f* outputs 1



f* covers *g

f=1 whenever g=1

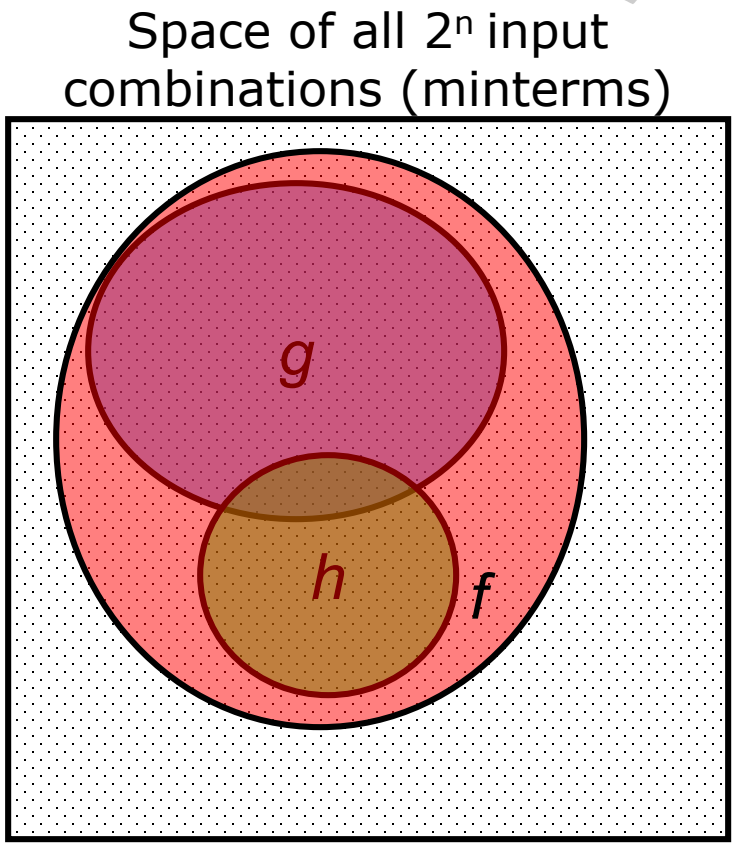
$$f \geq g$$

Implicants

= input combinations for which *g* outputs 1

= input combinations for which *f* outputs 1

= input combinations for which *h* outputs 1

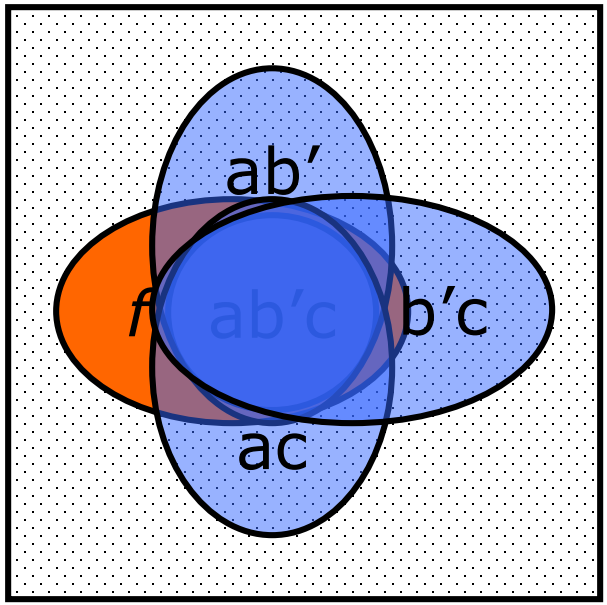


If *g* is a product term & $g \leq f$, Then *g* is an **implicant** of *f*.

Prime Implicants

- Removing a literal from any product term (any implicant) makes it cover twice as many minterms.
  - Removing a literal "grows" the term
  - ex. 3 variables: ab'c covers 1 minterm

    ab'
    ac
    b'c
    } each cover 2 minterms



ab'c is an implicant of *f*.

Any way of removing a literal makes ab'c no longer imply *f*. So ab'c is a **prime implicant** of *f*.

Implicant: Any product term that *implies* a function F (i.e., if, for some input combination, product term P = 1, then F = 1 for the same input combination)

YZ \ WX	00	01	11	10
00				
01	1	1		
11	1	1	1	
10			1	1

These are only a few of the implicants of this function…

Prime Implicant: An implicant such that if one literal is removed, the resulting product term no longer implies F

YZ \ WX	00	01	11	10
00				
01	1	1		
11	1	1	1	
10			1	1

Essential Prime Implicant: A prime implicant that covers a minterm that is not covered by any other prime implicants

YZ \ WX	00	01	11	10
00				
01	1	1		
11	1	1	1	
10			1	1

Theorem: The minimal SOP of a function is a sum of prime implicants

Tabular Generation of Prime Implicants (Quine-McCluskey Procedure)

- Main Theorems:
  - Adjacency: $x' \, p + x \, p = p$ (create "larger" implicants)
  - Absorption: $p + x \, p = p$ (delete subsumed implicants)
- Procedure:
  - Arrange product terms in groups such that all terms in one group have the same number of 1s in their binary representation, their Group Index
  - Starting from minterms, arrange groups in ascending-index order
  - Apply adjacency theorem to product terms from adjacent groups only
  - Remove subsumed product terms using absorption theorem
- Product term representation: $p = p_n p_{n-1} \ldots p_2 p_1$
  where

$$p_i = \begin{cases} 0 & \text{if ith variable appears complemented} \\ 1 & \text{if ith variable appears uncomplemented} \\ - & \text{if ith variable does not appear} \end{cases}$$

QM Example
Generation of Prime Implicants

$$F = \sum_{A,B,C,D,E}(1,3,15,17,19,29,31)$$

index	minterm	A	B	C	D	E
1	1	0	0	0	0	1
2	3	0	0	0	1	1
	17	1	0	0	0	1
3	19	1	0	0	1	1
4	15	0	1	1	1	1
	29	1	1	1	0	1
5	31	1	1	1	1	1

implicant	A	B	C	D	E

QM Example
Generation of Prime Implicants

$$F = \sum_{A,B,C,D,E}(1,3,15,17,19,29,31)$$

implicant	A	B	C	D	E
√ 1,3	0	0	0	-	1
√ 1,17	-	0	0	0	1
√ 3,19	-	0	0	1	1
√17,19	1	0	0	-	1
P_1: 15,31	-	1	1	1	1
P_2: 29,31	1	1	1	-	1

implicant	A	B	C	D	E
P_3:1,3,17,19	-	0	0	-	1

$$P_1 = BCDE \quad P_2 = ABCE \quad P_3 = B'C'E$$

QM Example
Minimization by Set Covering

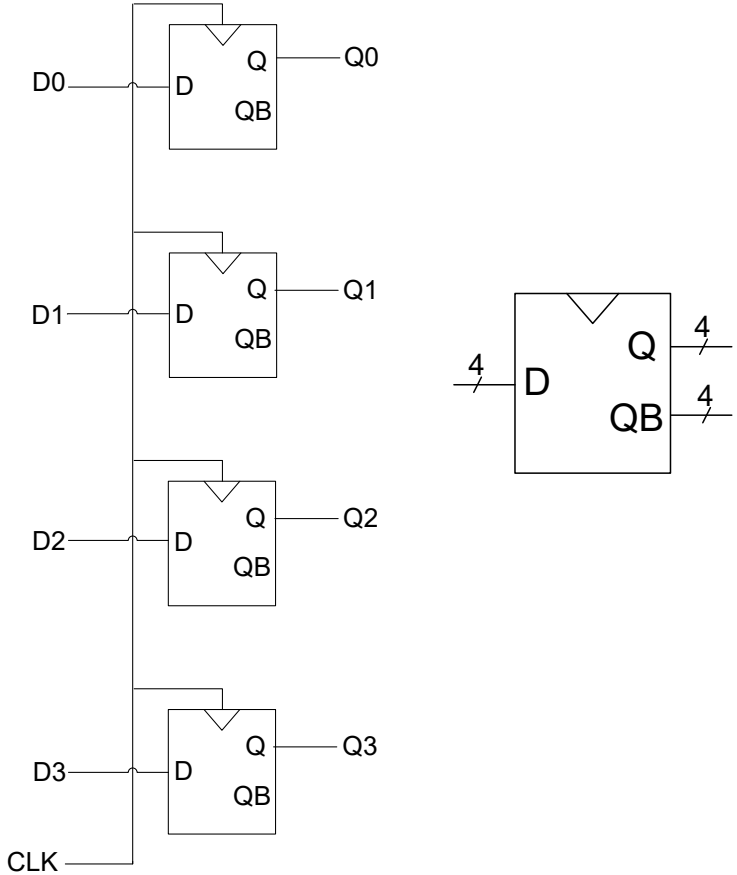
$$F = \sum_{A,B,C,D,E}(1,3,15,17,19,29,31)$$

PIs (Variables)

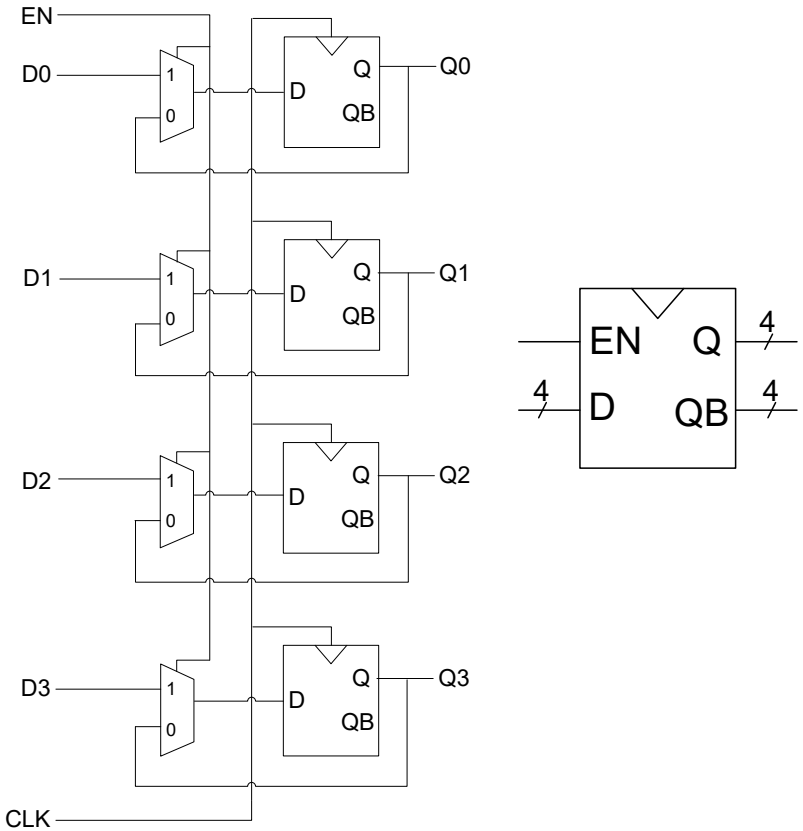
Minterms (Constraints)	P_1 $BCDE$	P_2 $ABCE$	P_3 $B'C'E$
1			1
3			1
15	1		
17			1
19			1
29		1	
31	1	1	

Registers

- A collection of two or more D flip-flops with a common clock is called a **register**
- Used to store a collection of bits



- Adding a mux at the input of each D FF allows for loading new value or storing current value



Parallel Counter

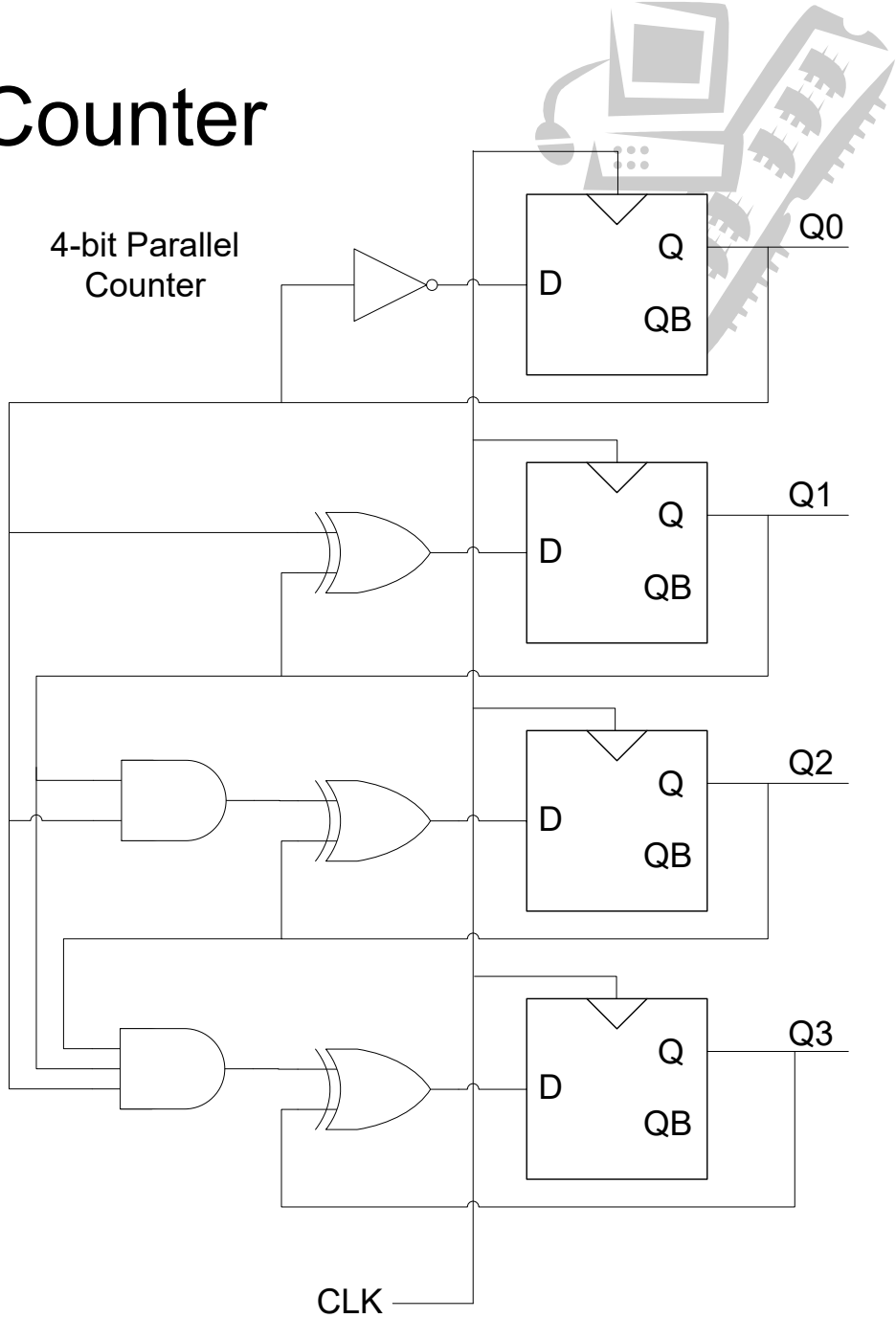
- Would like *all* state variables to run on the same clock signal
- Examining the binary code yields an easy implementation:

b_2	b_1	b_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

- Under what condition does each bit toggle?

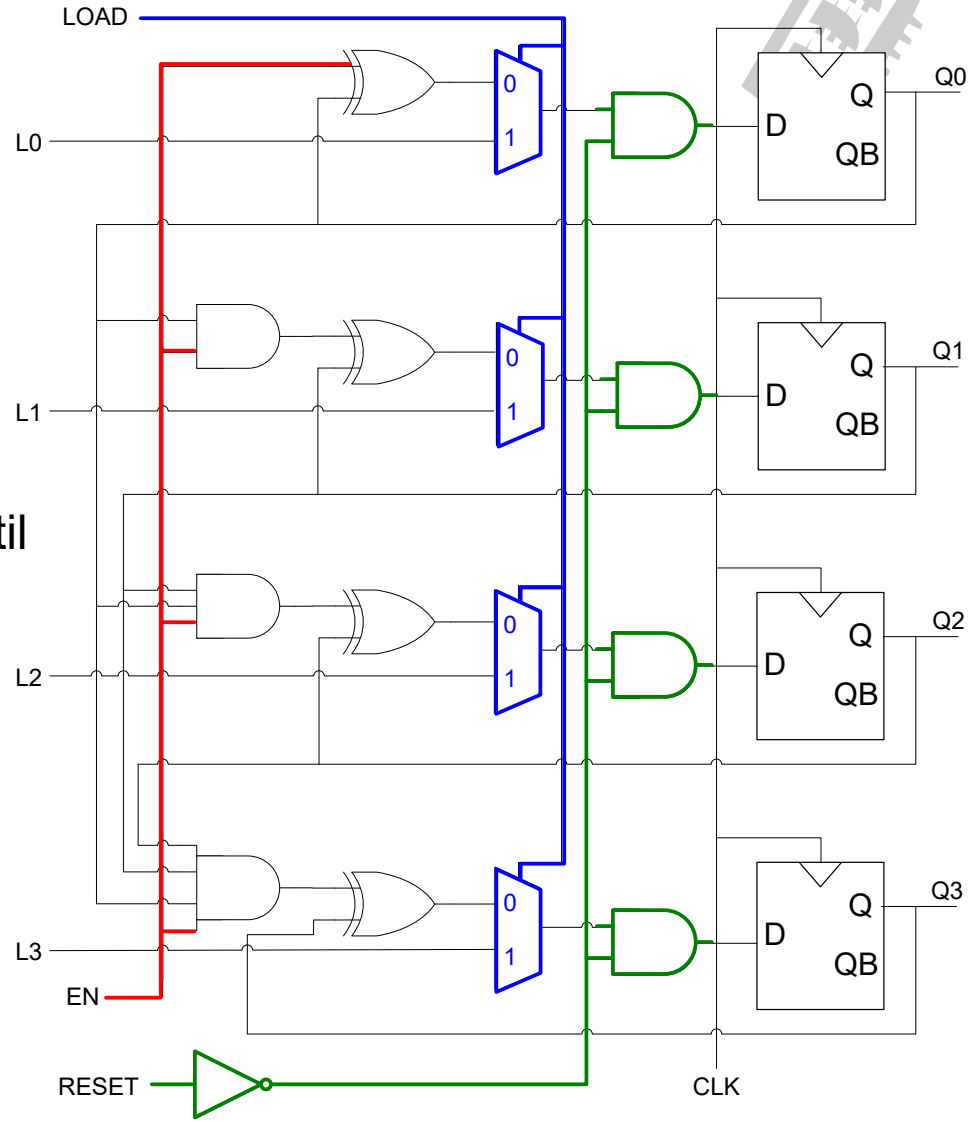
- *When all bits of lesser significance are 1!*

- This is true no matter how many bits in the counter



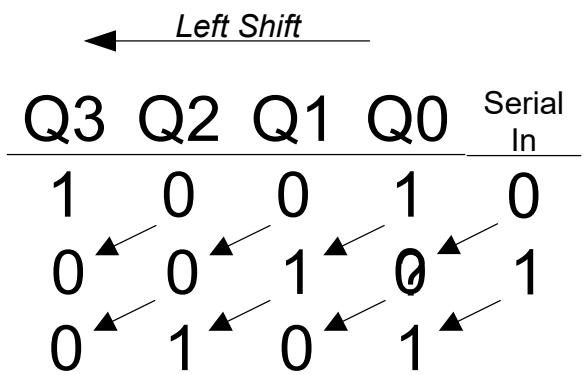
Parallel Counter with Enable, Load, and Reset

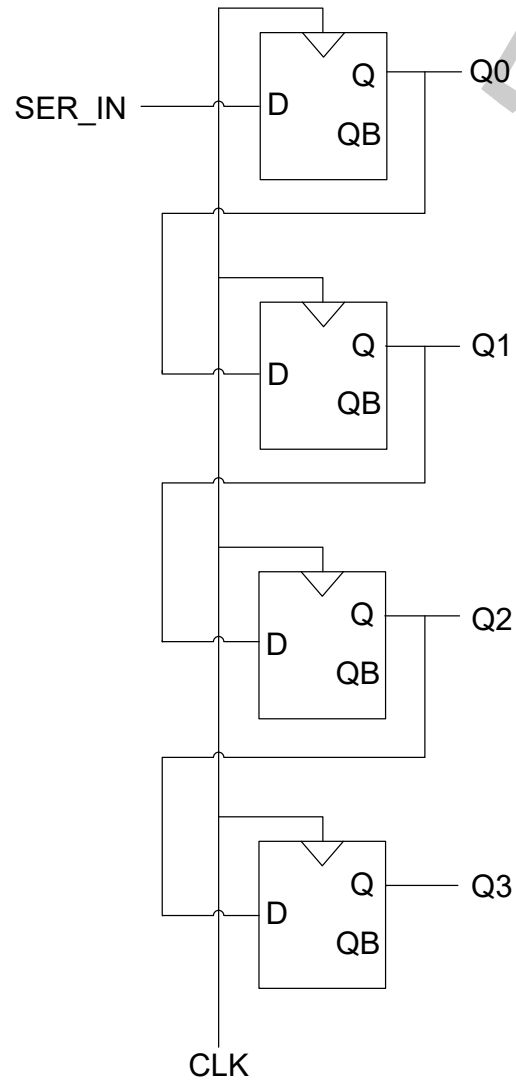
- New counter inputs
  - **RESET**: reset all state variables to 0
  - **ENABLE**: enable = 1 → count, enable = 0 → hold
  - **LOAD**: set state variables to load input values
- New inputs are *synchronous*
  - Their effects are not seen until the clock edge



Shift Registers

- A **shift register** allows for the shifting of its bits by one bit position on every clock edge

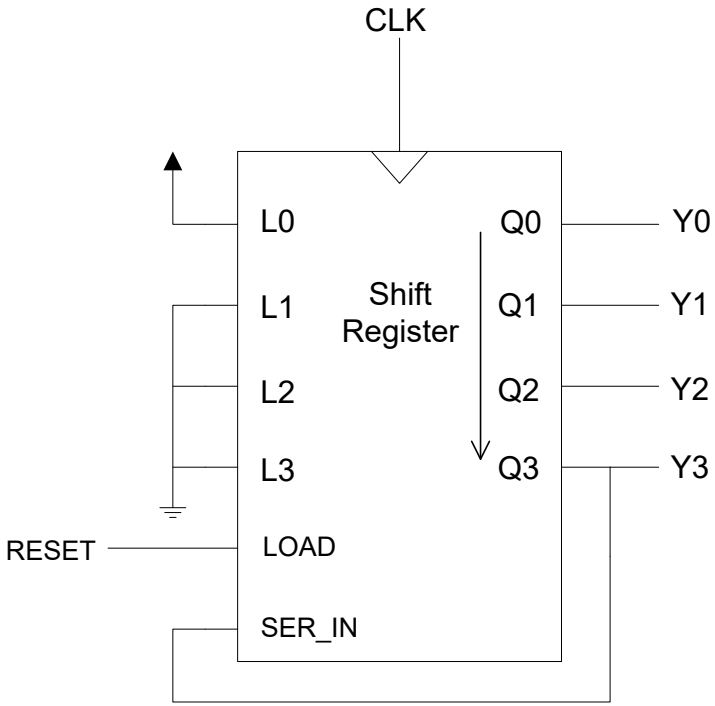


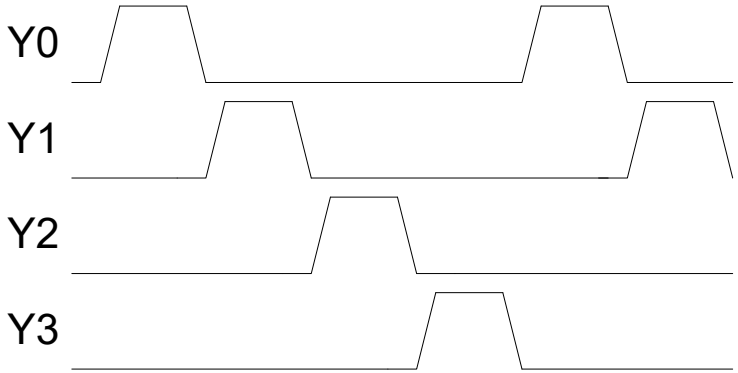


Using Shift Registers as Counters

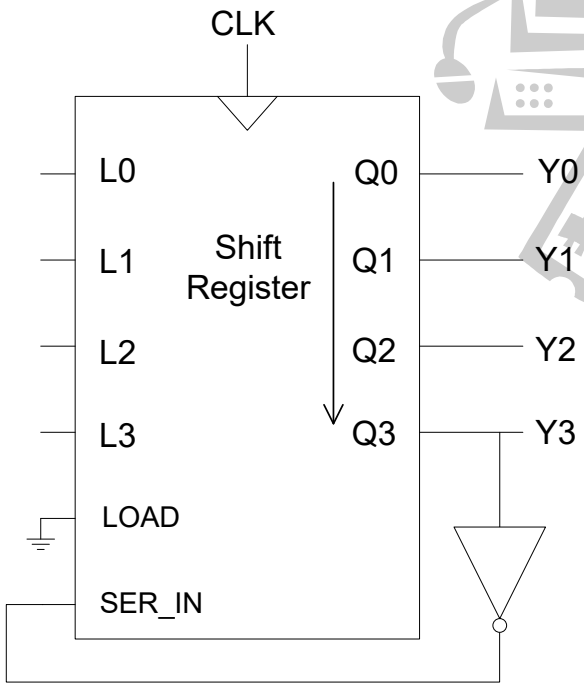
- *n*-bit **ring counter**
  - Implemented with an *n*-bit shift register with the serial out bit fed into the serial in input
  - Counts through sequence of *n* one-hot encodings

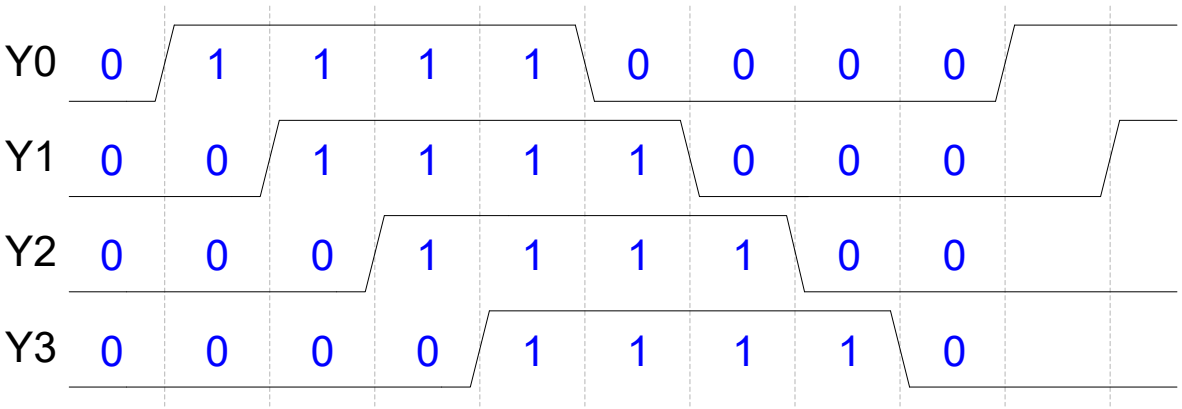
RESET	Y0	Y1	Y2	Y3
1	x	x	x	x
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
0	1	0	0	0





- *n*-bit **Johnson counter**
  - Implemented with an *n*-bit shift register with the complement of the serial out bit fed into the serial in input





An n-bit Johnson Counter has 2n states in its counting sequence

Data vs. Control State

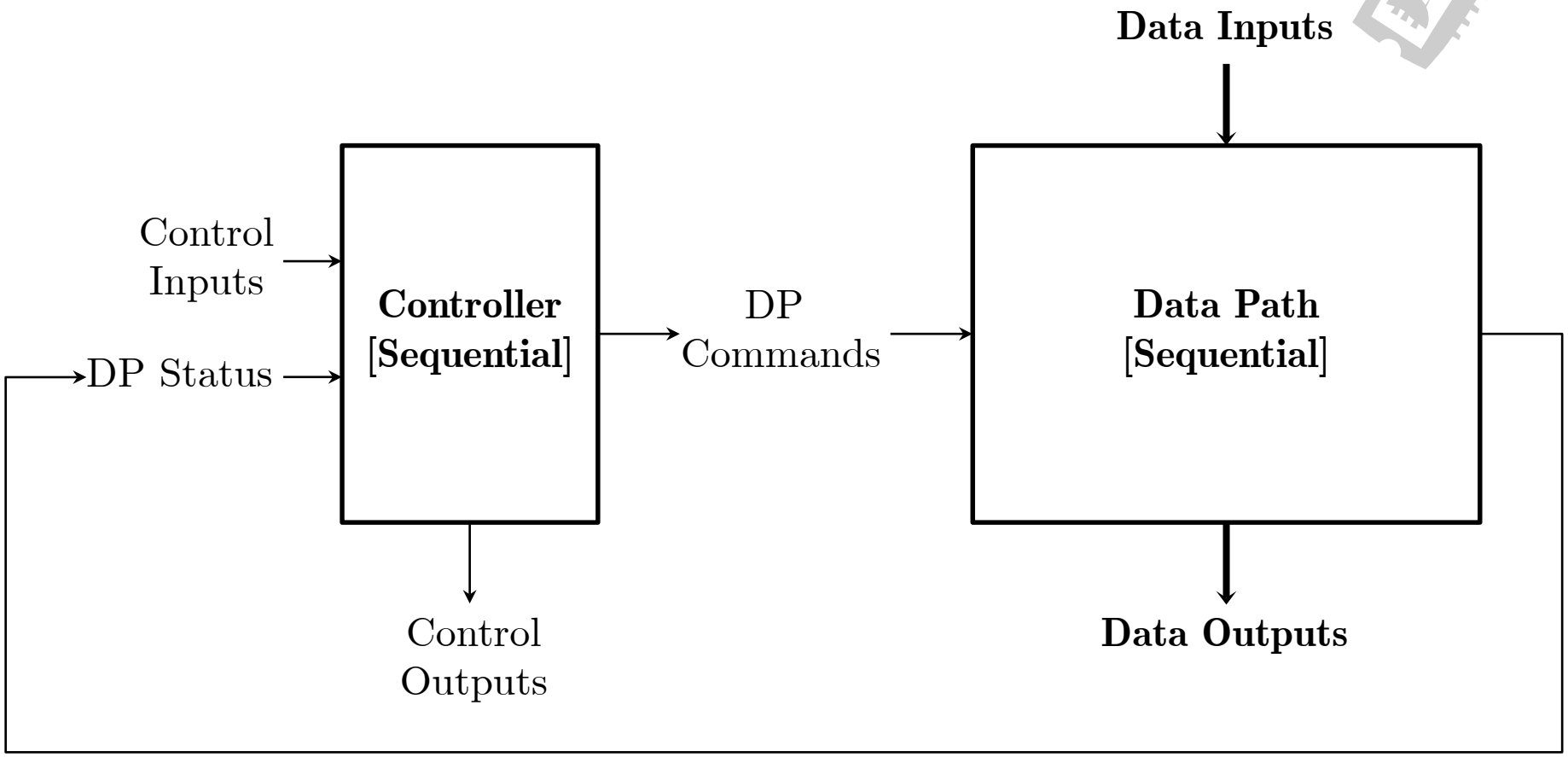
- Not all bits are equal!
- Some bits have meaning only in the context of other bits, e.g.,
  - Positional weight (numbers)
  - Codes (ASCII, Unicode, Colors, …)
- Multi-bit “words” represent data
- Single bits represent control
- Data state >> control state (think 64-bit numbers)
- Encoding sequential circuits that involve data at the single-bit level:
  - Leads to the so-called state explosion,
  - Loses the semantics (meaning) of data, and is
  - Unnecessary

Register Transfer Level (RTL)

- **Datapath**: Computation on “wide” signals
  - Arithmetic (Add, Subtract, Multiply, Count, etc.)
  - Logical (Shift right/left, Arith Shift, bit-wise, etc.)
  - Other (Clear, load, hold)

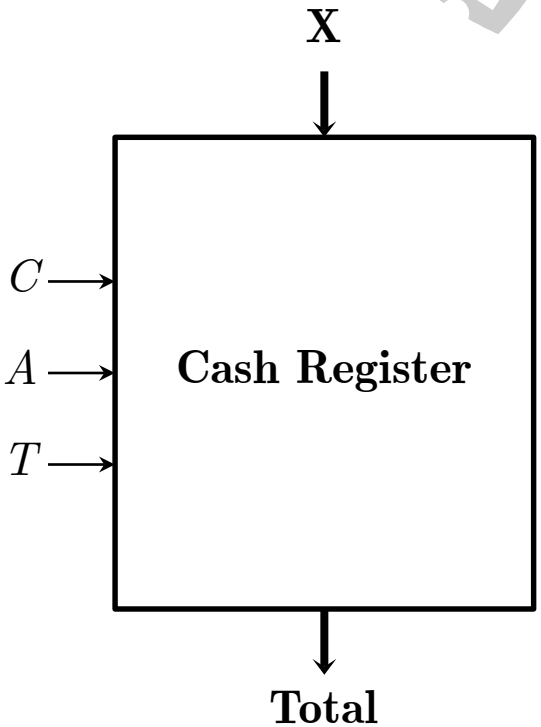
- **Controller**: Orchestrating Datapath operations

Structure of RTL Sequential Circuit



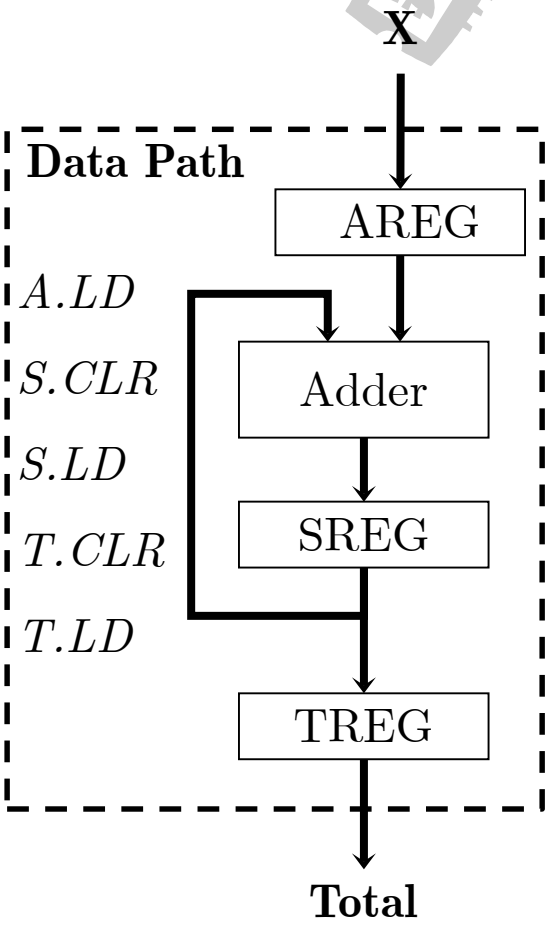
A “Cash Register”

- Function:
  Compute total price of a number of items
- Inputs:
  - **X**: W-bit unsigned integer
  - C: Clear
  - A: Add
  - T: Total
- Output:
  - **Total**: W-bit sum of added numbers
- Spec:
  - $C = 1$ clears **Total**
  - $A = 1$ adds next **X**
  - $T = 1$ displays **Total**
  - C, A, and T are one-hot

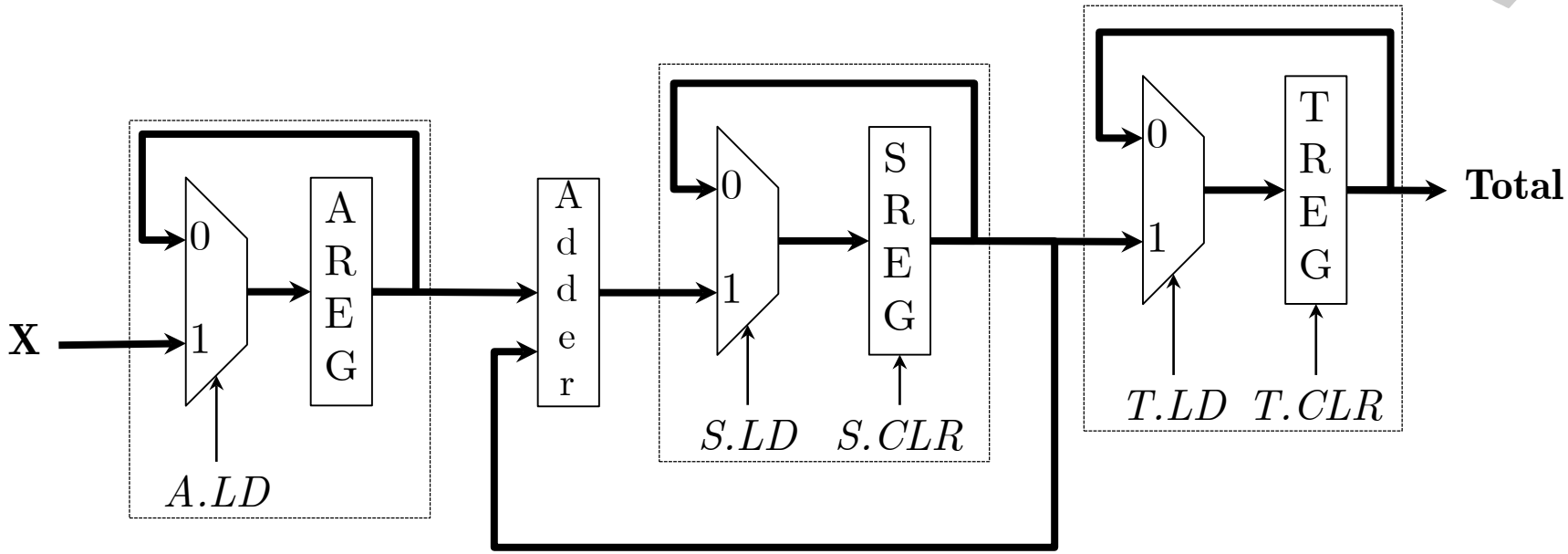


Designing the Datapath

- Required Components:
  - Register to store input **X**: AREG
  - Register to store output **Total**: TREG
  - Register to store intermediate sums: SREG
  - Adder
- Component functionality:
  - AREG: Load ($A.LD$)
  - TREG: Load ($T.LD$) and Clear ($T.CLR$)
  - SREG: Load ($S.LD$) and Clear ($S.CLR$)
- Datapath Architecture:

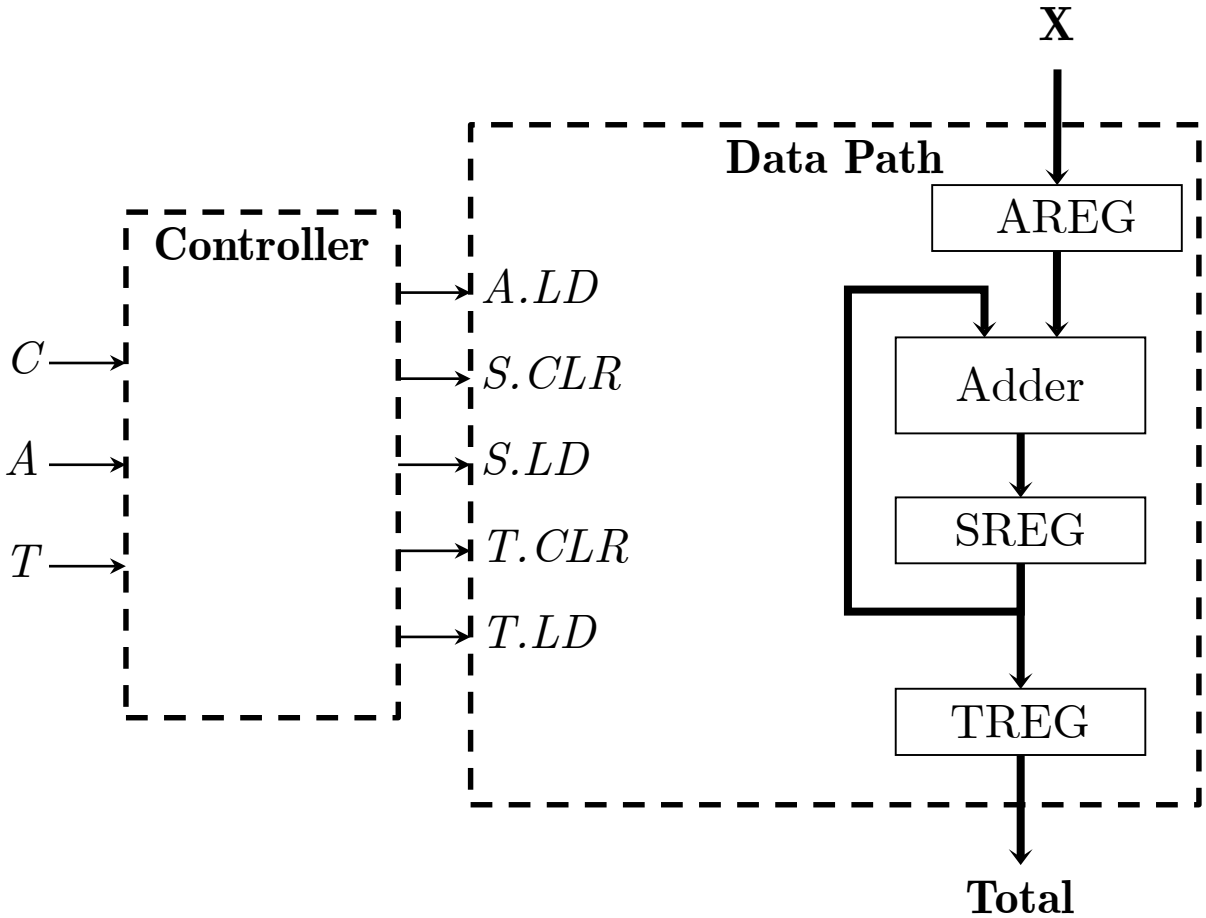


Detailed Datapath

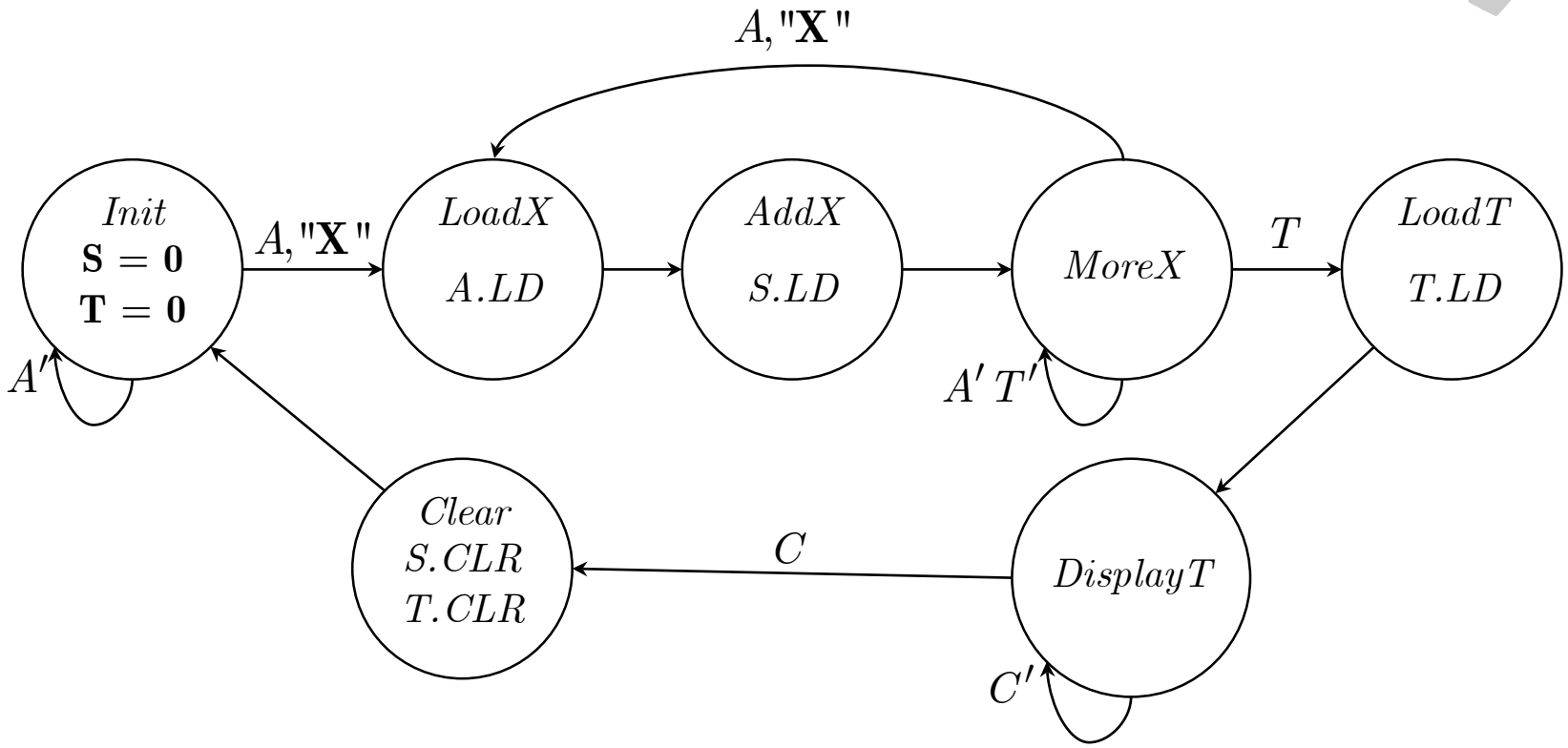


$$\mathbf{A}^{+} = A.LD\,?\,\mathbf{X} : \mathbf{A} \qquad \mathbf{S}^{+} = S.CLR\,?\,\mathbf{0} : \Big(S.LD\,?\,\big(\mathbf{S}+\mathbf{A}\big) : \mathbf{S}\Big) \qquad \mathbf{T}^{+} = T.CLR\,?\,\mathbf{0} : \Big(T.LD\,?\,\mathbf{S} : \mathbf{T}\Big)$$

Designing the Controller

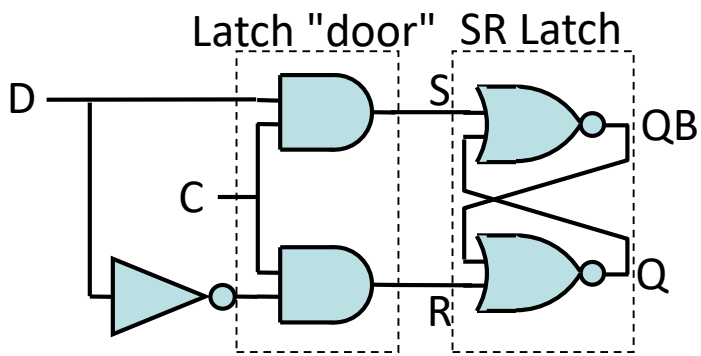


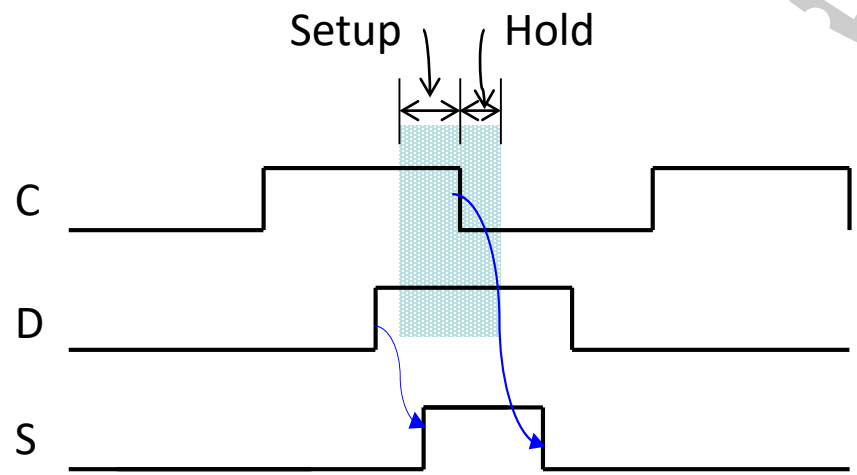
Controller State Diagram



Setup and Hold Times

Positive Level-Sensitive D Latch



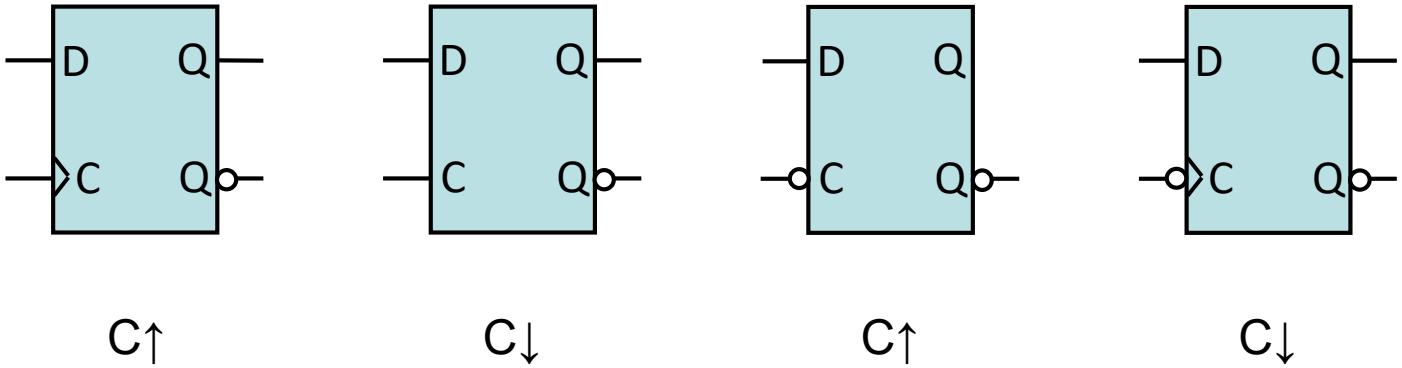


- To set, S must be held at 1 for at least 2 NOR delays
- S = D & C
- D must change to 1 at least 2 NOR delays before C goes to 0 (closes the latch)

Setup and Hold Times

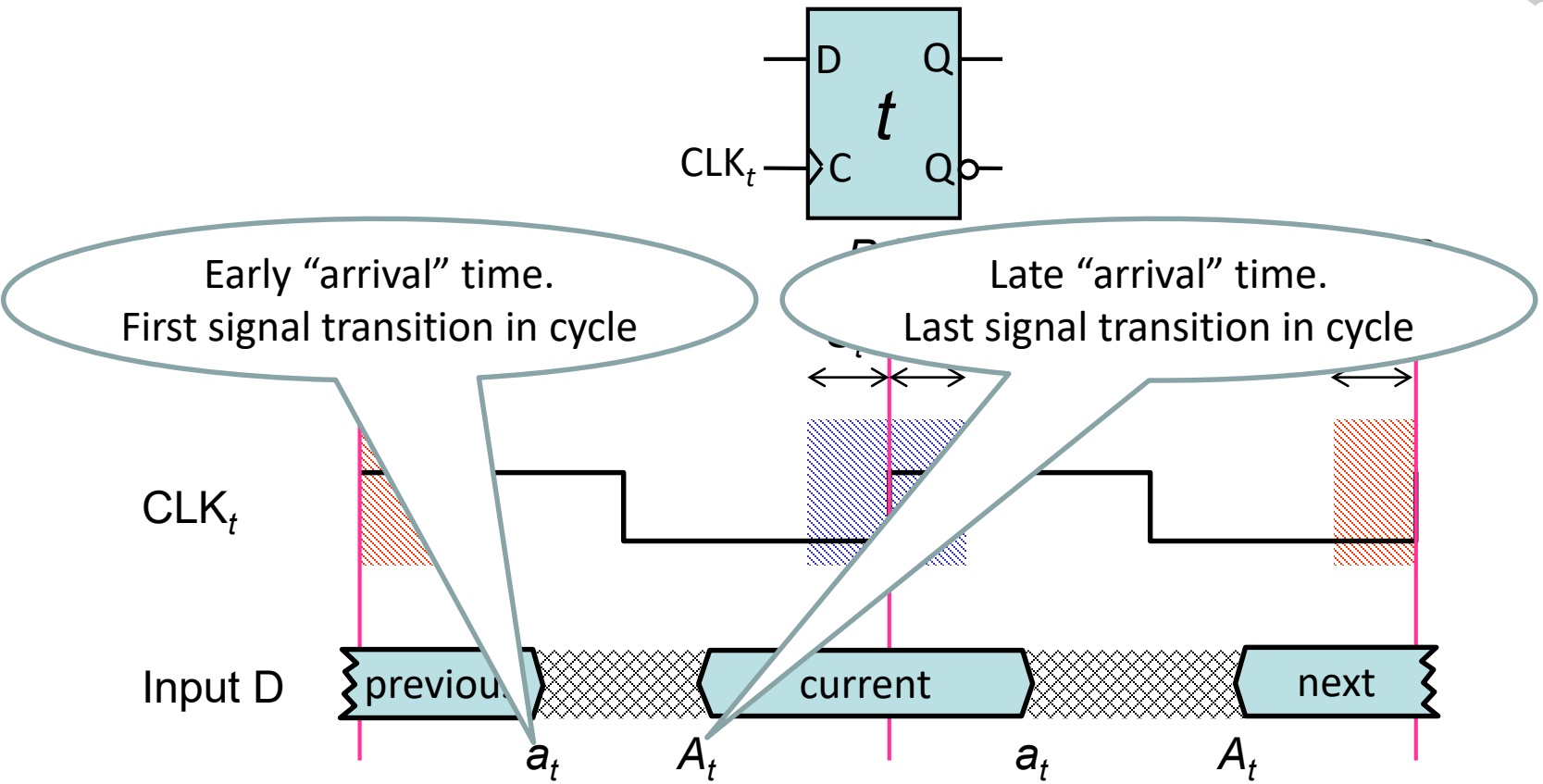
- By insuring that the D input is stable for a specified minimum length of time before (setup) and after (hold) the **appropriate clock edge** we eliminate metastability!
- Assume that setup and hold times are provided. They can be calculated, but the analysis is tricky.
- Which clock edge?

  Edge that "closes" the latch



Setup and Hold Constraints

Positive Edge-Triggered D Flip-Flop



Hold Requirement

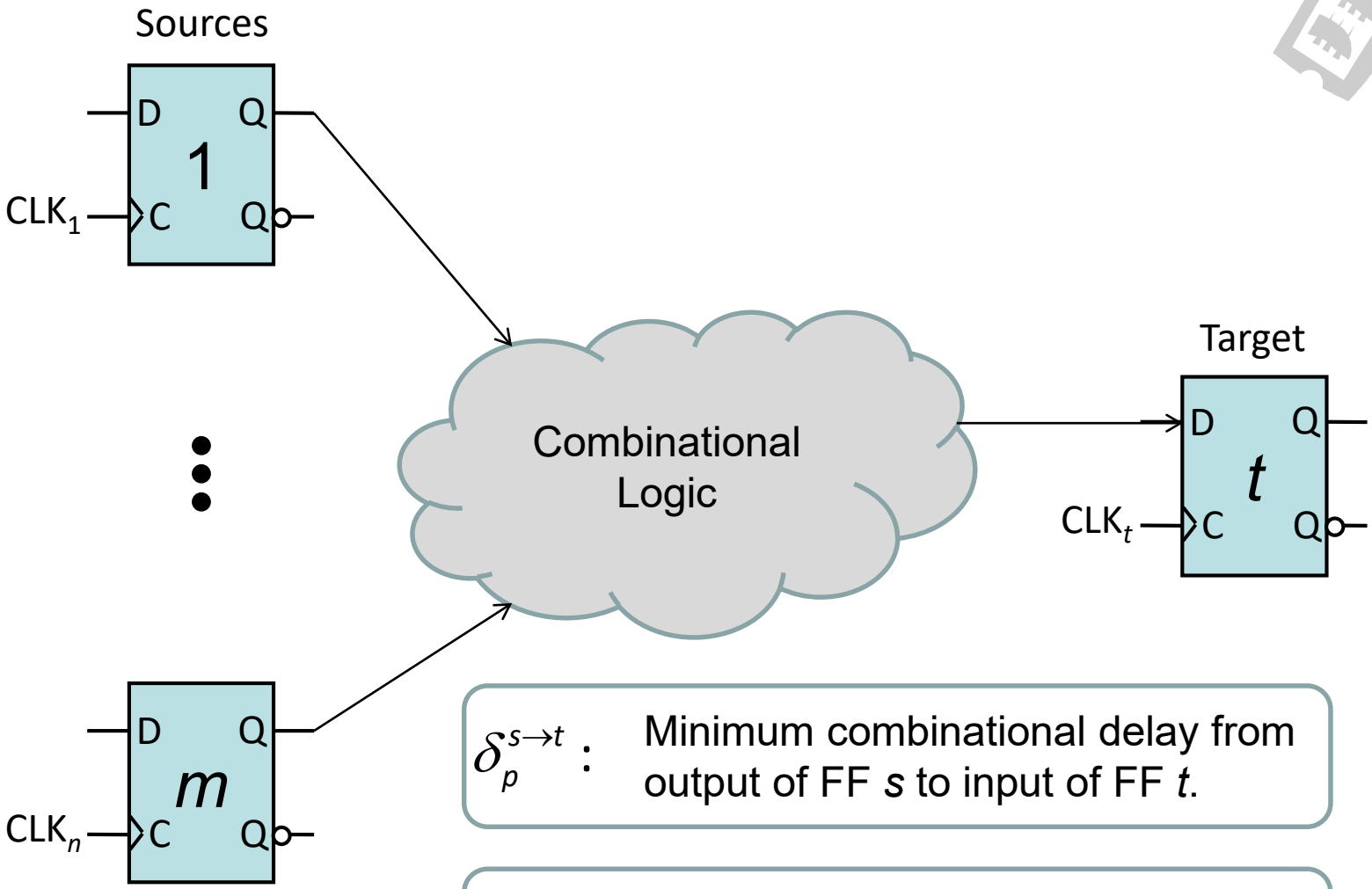
$$a_t \geq H_t$$

Setup Requirement

$$A_t \leq P - S_t$$

$$0 \leq a_t, A_t \leq P$$

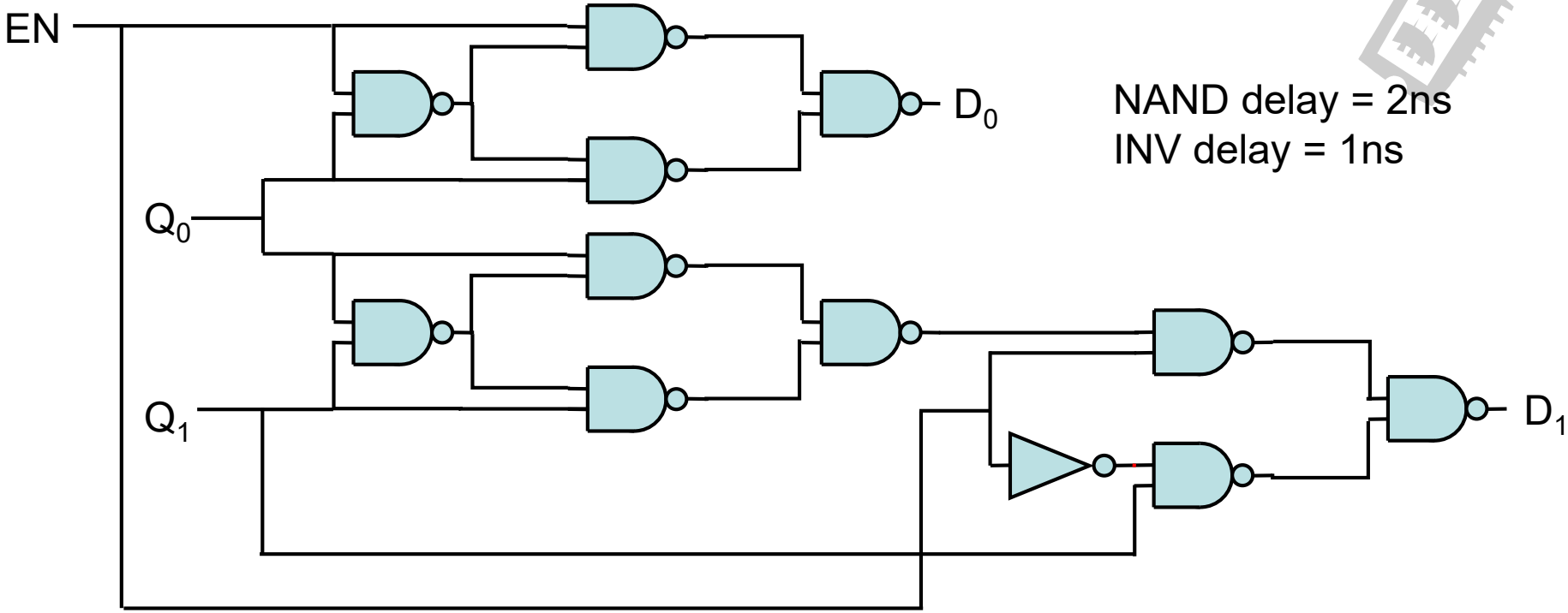
Combinational Delay Model



$\delta_p^{s \to t}$: Minimum combinational delay from output of FF *s* to input of FF *t*.

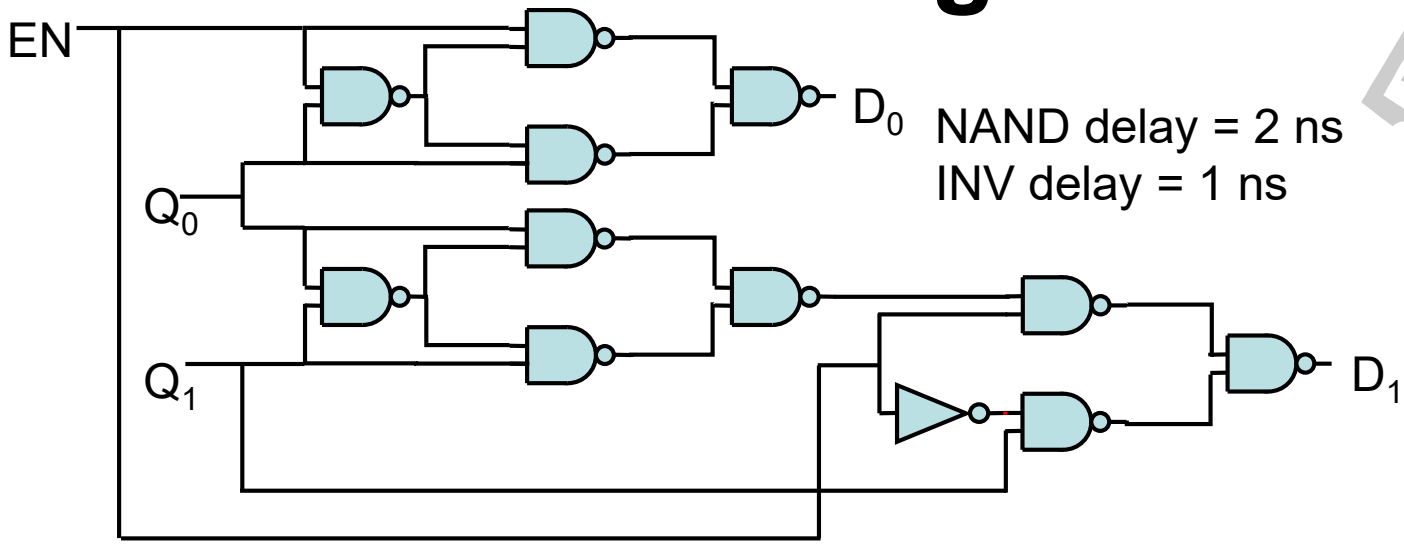
$\Delta_p^{s \to t}$: Maximum combinational delay from output of FF *s* to input of FF *t*.

Gate Implementation of Example



	D_0	D_1
EN	$[\delta_p^{EN\to D_0}, \Delta_p^{EN\to D_0}]$	$[\delta_p^{EN\to D_1}, \Delta_p^{EN\to D_1}]$
Q_0	$[\delta_p^{Q_0\to D_0}, \Delta_p^{Q_0\to D_0}]$	$[\delta_p^{Q_0\to D_1}, \Delta_p^{Q_0\to D_1}]$
Q_1	$[\delta_p^{Q_1\to D_0}, \Delta_p^{Q_1\to D_0}]$	$[\delta_p^{Q_1\to D_1}, \Delta_p^{Q_1\to D_1}]$

Combinational Timing Summary



NAND delay = 2 ns
INV delay = 1 ns

From *EN*

$[\delta_p^{EN\to D_0}, \Delta_p^{EN\to D_0}] = [4,6]$

$[\delta_p^{EN\to D_1}, \Delta_p^{EN\to D_1}] = [4,5]$

From Q_0

$[\delta_p^{Q_0\to D_0}, \Delta_p^{Q_0\to D_0}] = [4,6]$

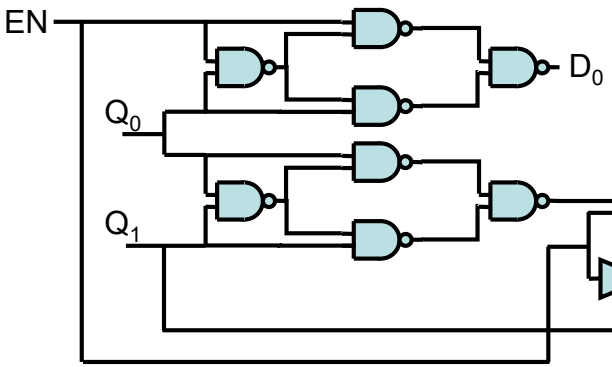
$[\delta_p^{Q_0\to D_1}, \Delta_p^{Q_0\to D_1}] = [8,10]$

From Q_1

$[\delta_p^{Q_1\to D_1}, \Delta_p^{Q_1\to D_1}] = [4,10]$

	D_0	D_1
EN	[4,6]	[4,5]
Q_0	[4,6]	[8,10]
Q_1	[∞, ∞]	[4,10]

Detailed Timing Analysis



Clock-to-Q delay = 4 ns
Setup time = 4 ns
Hold time = 2 ns

	D_0	D_1
~~EN~~	~~[4,6]~~	~~[4,5]~~
Q_0	[4,6]	[8,10]
Q_1	$[\infty,-\infty]$	[4,10]

Departure times (from FFs): $d_0 = D_0 = d_1 = D_1 = t_p^{C\to Q} = 4$ ns

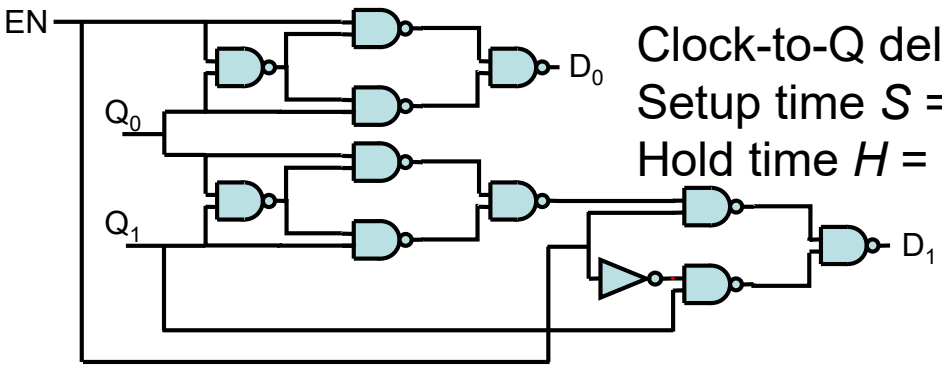
$$a_0 = \min\left(t_p^{C\to Q} + \delta_P^{Q_0\to D_0}, t_p^{C\to Q} + \delta_p^{Q_1\to D_0}\right)$$
$$= \min\left(4+4, 4+\infty\right)$$
$$= 8 \text{ ns}$$

$$A_0 = \max\left(t_p^{C\to Q} + \Delta_p^{Q_0\to D_0}, t_p^{C\to Q} + \Delta_p^{Q_1\to D_0}\right)$$
$$= \max\left(4+6, 4+-\infty\right)$$
$$= 10 \text{ ns}$$

$$a_1 = \min\left(t_p^{C\to Q} + \delta_p^{Q_0\to D_1}, t_p^{C\to Q} + \delta_p^{Q_1\to D_1}\right)$$
$$= \min\left(4+8, 4+4\right)$$
$$= 8 \text{ ns}$$

$$A_1 = \max\left(t_p^{C\to Q} + \Delta_p^{Q_0\to D_1}, t_p^{C\to Q} + \Delta_p^{Q_0\to D_1}\right)$$
$$= \max\left(4+10, 4+10\right)$$
$$= 14 \text{ ns}$$

Detailed Timing Analysis (Cont'd)



Clock-to-Q delay = 4 ns
Setup time S = 4 ns
Hold time H = 2 ns

	D_0	D_1
~~EN~~	~~[4,6]~~	~~[4,5]~~
Q_0	[4,6]	[8,10]
Q_1	$[\infty,-\infty]$	[4,10]

$a_0 = 8$ ns

$a_1 = 8$ ns

Hold Requirement

$$a \geq H$$

$a_0 \geq H$ ✔

$a_1 \geq H$ ✔

$\therefore$ Hold requirements are met.

$A_0 = 10$ ns

$A_1 = 14$ ns

Setup Requirement

$$A \leq P - S \Rightarrow P \geq A + S$$

$P \geq A_0 + S = 10 + 4 = 14$ ns

$P \geq A_1 + S = 14 + 4 = 18$ ns

$\therefore P_{min} = 18$ ns

Unsigned vs. Signed Multiplication

$$P = Q \times M$$

Unsigned

$$M = 2^3 m_3 + 2^2 m_2 + 2^1 m_1 + 2^0 m_0$$

$$\begin{aligned} P = {} & Q \times 2^0 m_0 + \\ & Q \times 2^1 m_1 + \\ & Q \times 2^2 m_2 + \\ & Q \times 2^3 m_3 \end{aligned}$$

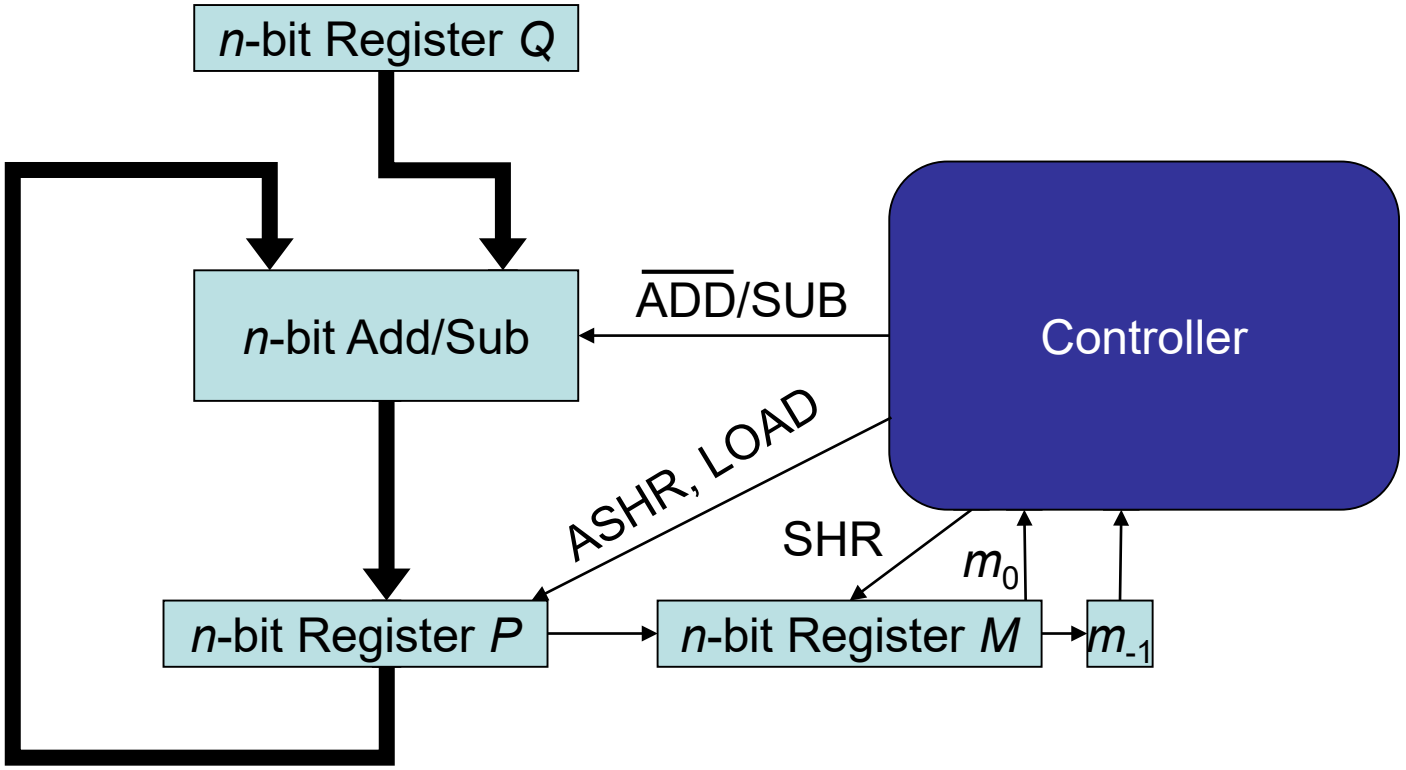
Signed

$$M = -2^3 m_3 + 2^2 m_2 + 2^1 m_1 + 2^0 m_0$$

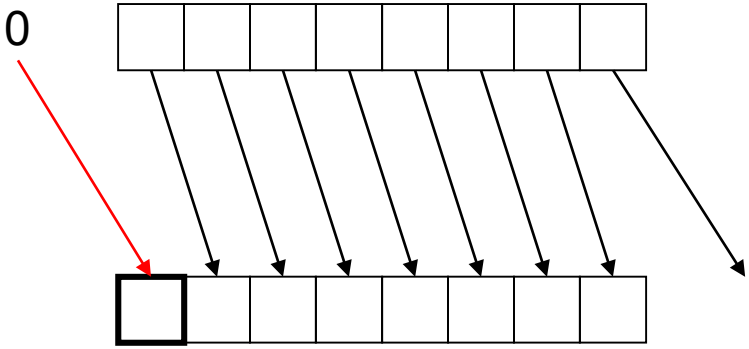
$m_{-1} = 0$

$$\begin{aligned} P = {} & Q \times 2^0 \left(m_{-1} - m_0\right) + \\ & Q \times 2^1 \left(m_0 - m_1\right) + \\ & Q \times 2^2 \left(m_1 - m_2\right) + \\ & Q \times 2^3 \left(m_2 - m_3\right) \end{aligned}$$

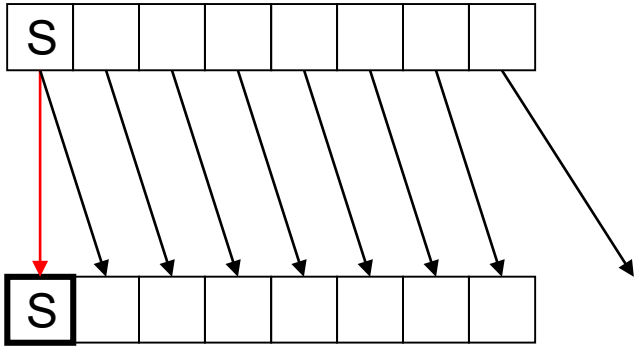
Sequential Multiplier V2.0



Logical vs. Arithmetic Right Shift



Logical Shift



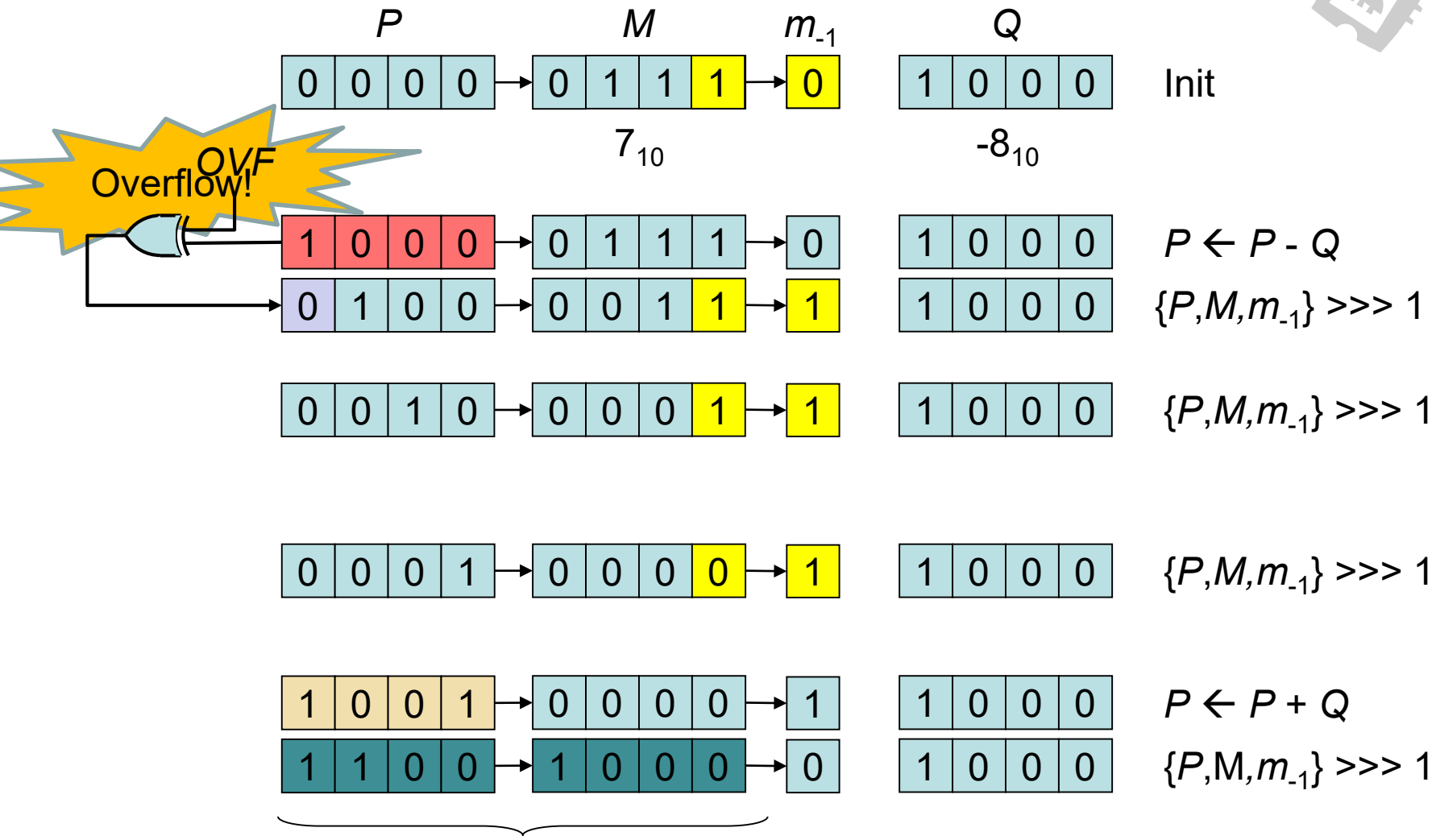
Arithmetic Shift

Multiplier V2.0 Execution Trace

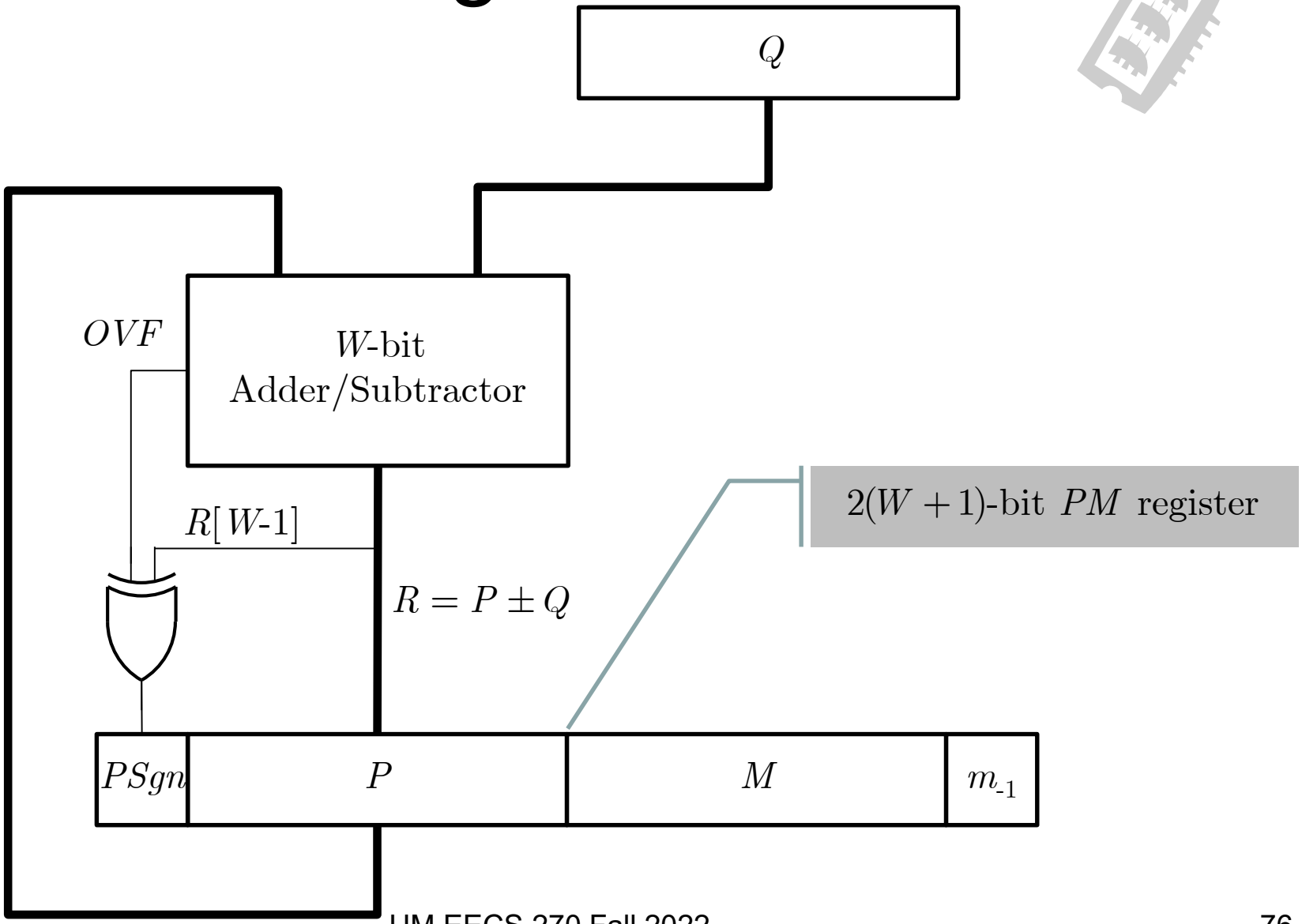
P	M	m_{-1}	Q	
0 0 0 0 →	0 1 1 1 →	0	1 0 1 0	Init
	7_{10}		-6_{10}	
0 1 1 0 →	0 1 1 1 →	0	1 0 1 0	$P \leftarrow P - Q$
0 0 1 1 →	0 0 1 1 →	1	1 0 1 0	$\{P,M,m_{-1}\}$ >>> 1
0 0 0 1 →	1 0 0 1 →	1	1 0 1 0	$\{P,M,m_{-1}\}$ >>> 1
0 0 0 0 →	1 1 0 0 →	1	1 0 1 0	$\{P,M,m_{-1}\}$ >>> 1
1 0 1 0 →	1 1 0 0 →	1	1 0 1 0	$P \leftarrow P + Q$
1 1 0 1 →	0 1 1 0 →	0	1 0 1 0	$\{P,M,m_{-1}\}$ >>> 1

-42_{10}

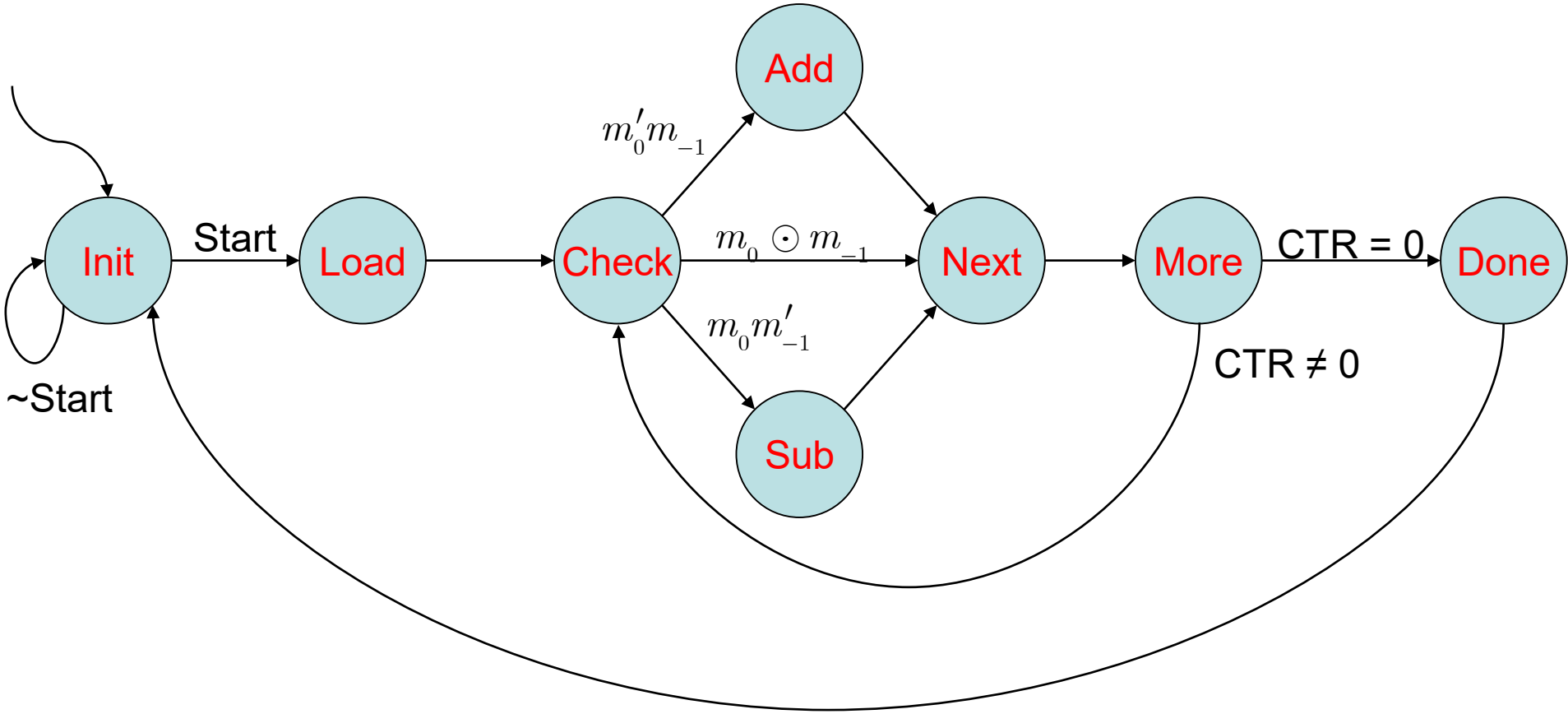
Multiplier V2.0 Execution Trace

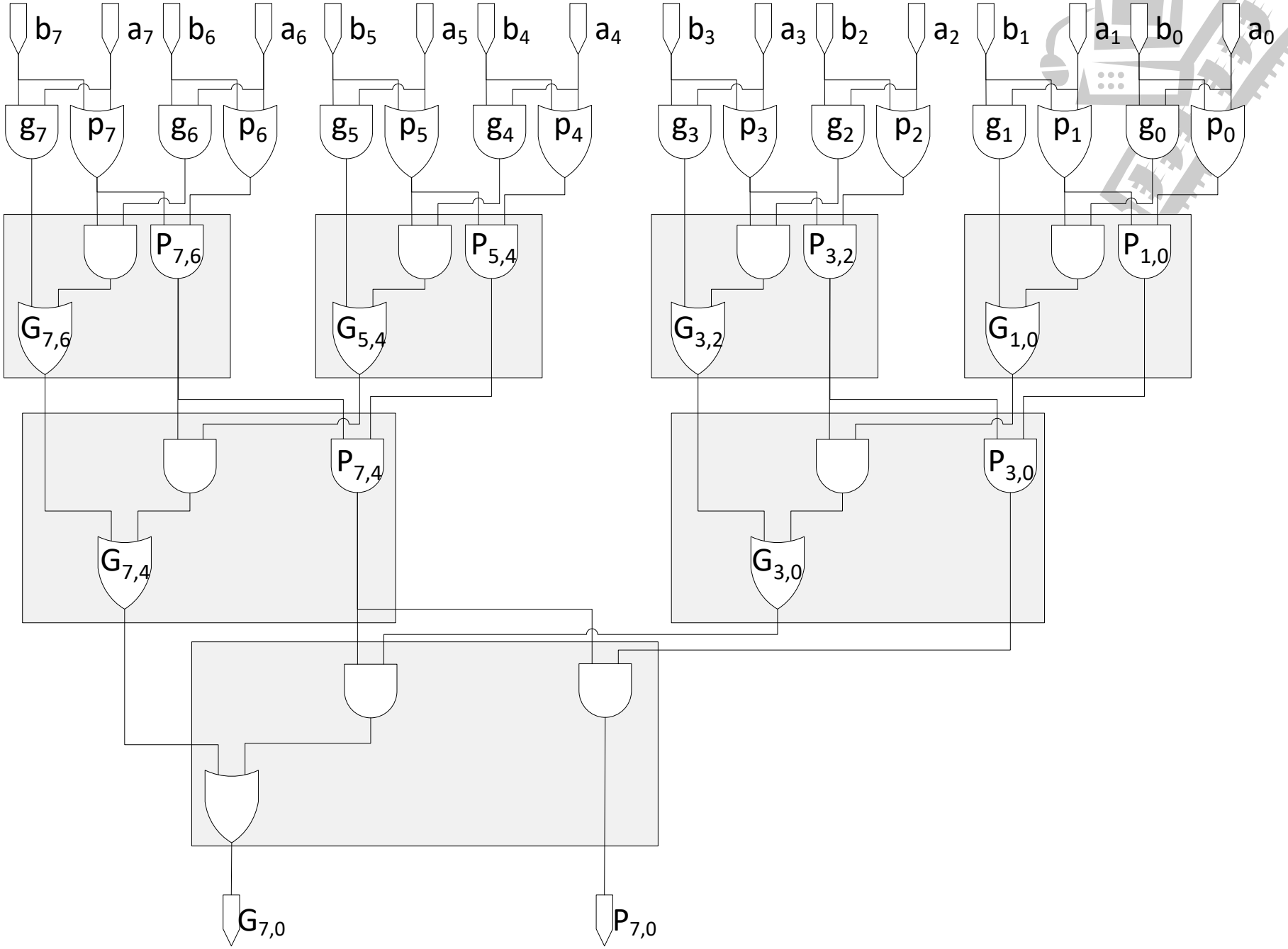


Correcting for Overflow



Controller State Diagram




b7
a7
b6
a6
b5
a5
b4
a4
b3
a3
b2
a2
b1
a1
b0
a0
g7
p7
g6
p6
g5
p5
g4
p4
g3
p3
g2
p2
g1
p1
g0
p0
P7,6
P5,4
P3,2
P1,0
G7,6
G5,4
G3,2
G1,0
P7,4
P3,0
G7,4
G3,0
G7,0
P7,0

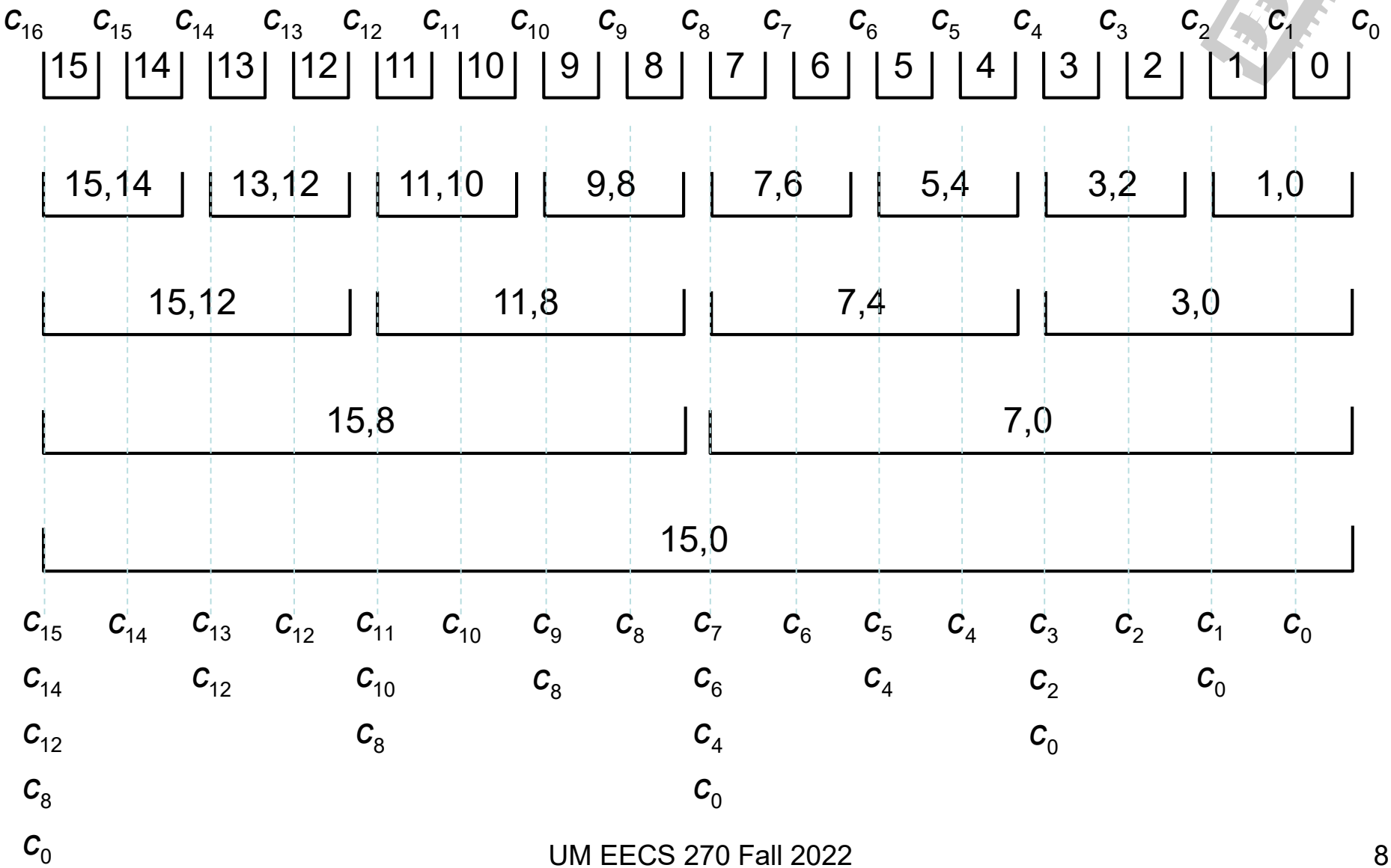
Generate/Propagate Equations

Depth(P,G)	(1, 1)	(2, 3)	(3, 5)	(4, 7)
$g_0 = a_0 b_0$				
$p_0 = a_0 + b_0$	$G_{1,0} = g_1 + p_1 g_0$			
$g_1 = a_1 b_1$	$P_{1,0} = p_1 p_0$			
$p_1 = a_1 + b_1$		$G_{3,0} = G_{3,2} + P_{3,2} G_{1,0}$		
$g_2 = a_2 b_2$		$P_{3,0} = P_{3,2} P_{1,0}$		
$p_2 = a_2 + b_2$	$G_{3,2} = g_3 + p_3 g_2$			
$g_3 = a_3 b_3$	$P_{3,2} = p_3 p_2$			
$p_3 = a_3 + b_3$			$G_{7,0} = G_{7,4} + P_{7,4} G_{3,0}$	
$g_4 = a_4 b_4$			$P_{7,0} = P_{7,4} P_{3,0}$	
$p_4 = a_4 + b_4$	$G_{5,4} = g_5 + p_5 g_4$			
$g_5 = a_5 b_5$	$P_{5,4} = p_5 p_4$			
$p_5 = a_5 + b_5$		$G_{7,4} = G_{7,6} + P_{7,6} G_{5,4}$		
$g_6 = a_6 b_6$		$P_{7,4} = P_{7,6} P_{5,4}$		
$p_6 = a_6 + b_6$	$G_{7,6} = g_7 + p_7 g_6$			
$g_7 = a_7 b_7$	$P_{7,6} = p_7 p_6$			
$p_7 = a_7 + b_7$				

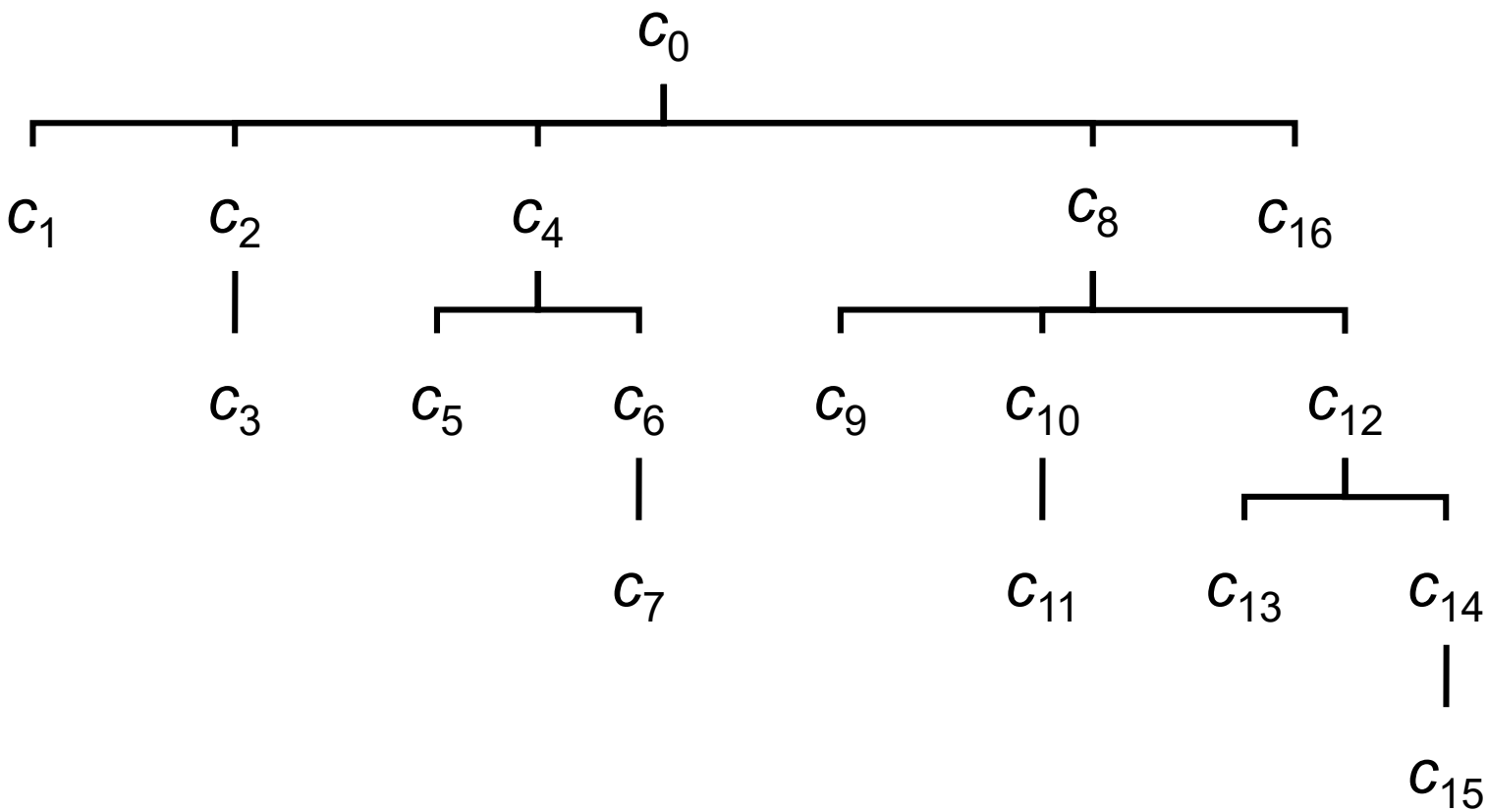
Delays in the GP "Parallel Prefix"

Level	Bit-Range Width	Δ_P	Δ_G
1	1	1	1
2	2	2	3
3	4	3	5
4	8	4	7
5	16	5	9

Options for Carry Computation

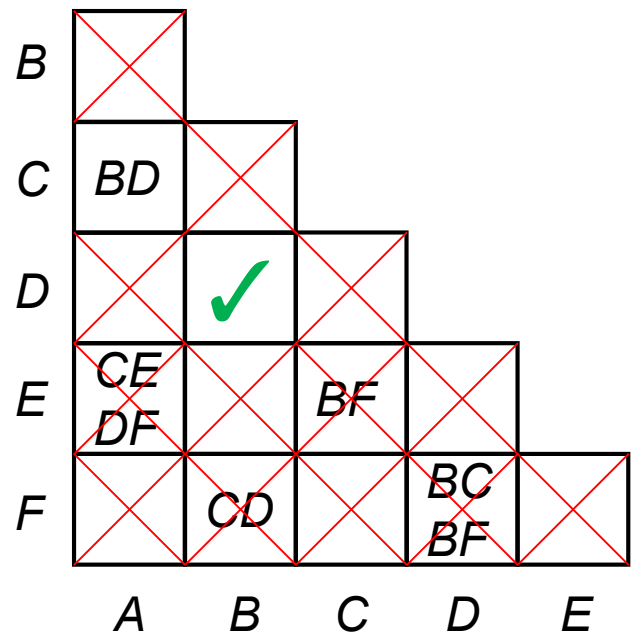


Carry Dependency Tree
(which carry propagates to which other carry)



Detecting Equivalent States
Merger Table

PS	NS, z	
	$x = 0$	$x = 1$
A	E,0	D,1
B	F,0	D,0
C	E,0	B,1
D	F,0	B,0
E	C,0	F,1
F	B,0	C,0



$\{(AC),(BD),(E),(F)\}$